Experts Live Denmark

The Future of AI-Driven Software Engineers

AI Master Class

Experts Live Denmark

The Future of AI-Driven Software Engineers

Thomas Martinsen

Tech Evangelist @ Twoday + Microsoft Regional Director & Microsoft AI MVP

Frederik Baun Hansen

Global Senior Architect @ Twoday

Henri Schulte

Partner Solution Architect @ Microsoft

Andreas Hoffmann

Software Developer @ Twoday

THANK YOU TO OUR AMAZING SPONSORS !

Guidelines for this master class

Be active and deliberate

Engage directly in the exercises and discussions rather than observing from the sidelines. The value of this master class comes from applying the concepts in practice, challenging assumptions, and testing ideas in real time. Approach the case work with intent: define goals clearly, articulate constraints, and avoid jumping to solutions before the problem is well understood.

Work with structure

Treat ideas, outputs, and decisions with an engineering mindset. Be explicit about assumptions, boundaries, and trade-offs. Approach problems methodically, and validate results instead of accepting them at face value. Use structure in how you think, collaborate, and document your work. Clear reasoning and disciplined execution are central to making AI-driven development reliable.

Connect to your reality

Continuously reflect on how the methods and patterns discussed apply to your own organization, systems, and responsibilities. Leverage the different perspectives in your team, especially across development and infrastructure roles. Aim to leave with concrete actions you can implement immediately, as well as a clearer understanding of what must change to adopt these practices effectively.

Takeaways for this master class

Engineering shifts from coding to intent and specs

Modern AI-native development moves the engineer's focus from writing code to defining intent, specifications, architecture boundaries, and validation gates. The spec becomes the core unit of work, enabling predictable and controlled agent execution.

Reliable AI development requires structure, not ad-hoc prompts

Professional AI-assisted development depends on structured prompt engineering, agent primitives, and context engineering. Treating prompts and agent behavior as reusable, versioned engineering assets makes outcomes predictable and scalable.

AI supports the full software lifecycle—not just code

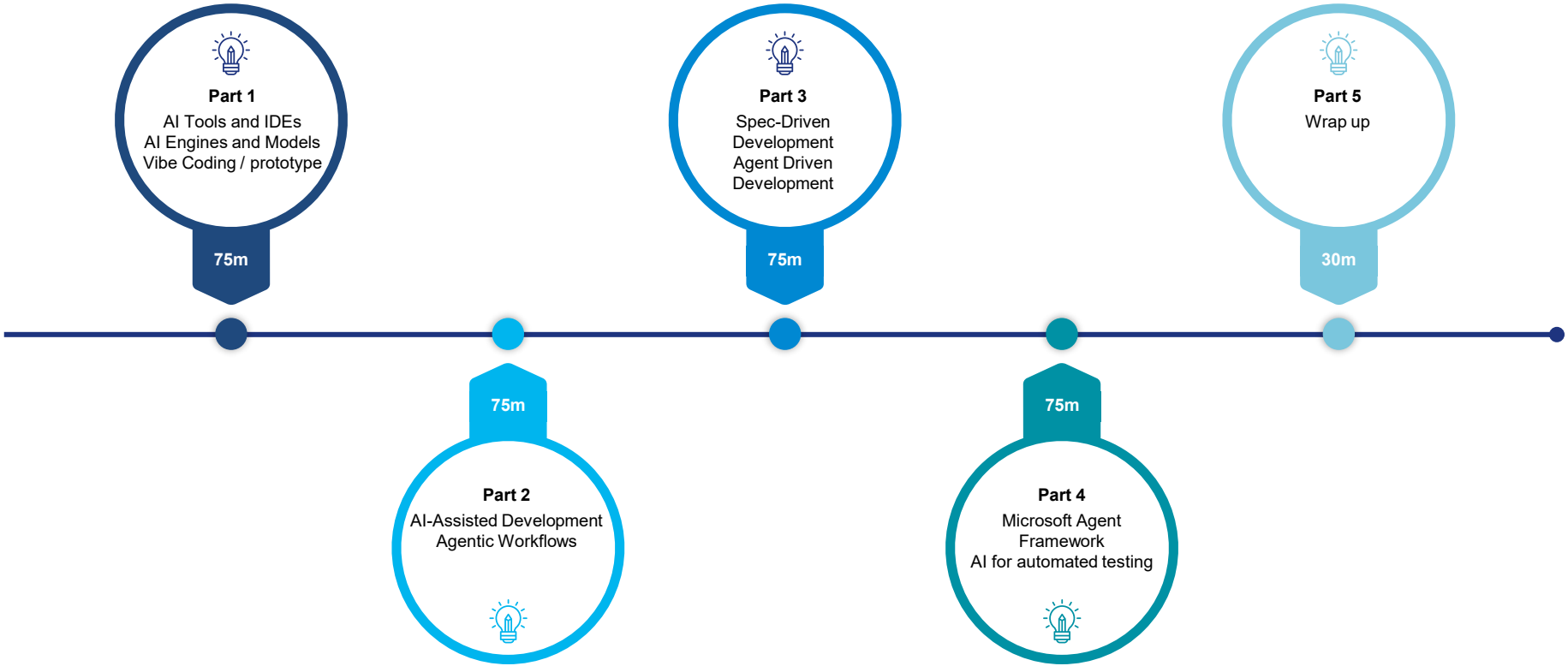
AI can do more than generate code: it can drive automated testing, documentation, validation, and other lifecycle tasks. These capabilities can be orchestrated end-to-end using interoperable frameworks like the Microsoft Agent Framework, with human oversight and validation gates.

GitHub Repos

https://github.com/ELDK26-AI-masterclass

Experts Live Denmark

Agenda




Threadly
SMART TEES, COOL STYLE
Custom Tees,
Powered by AI
Shop unique, smartly designed T-shirts.
Shop Now
EVOVING MINDS

Treadly Case Overview

AI Tools and Models

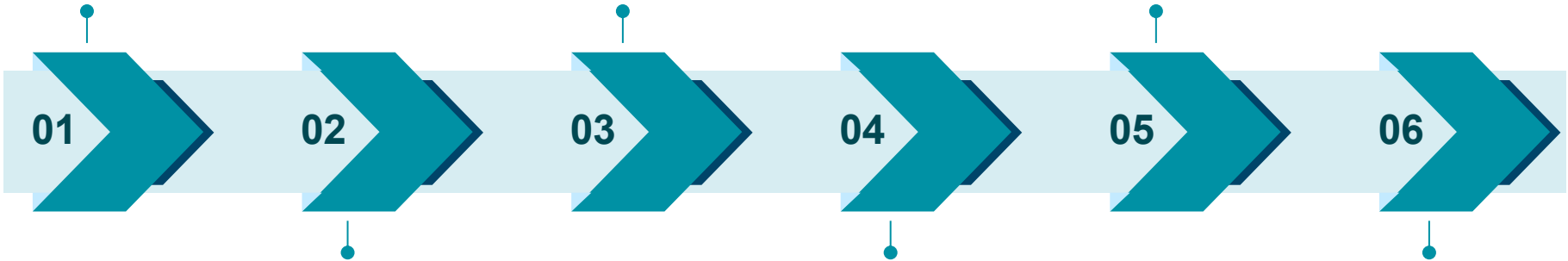
- Using AI tools and IDE integrations effectively
- Understanding model strengths and limitations

AI-Assisted Development

- Prompt engineering beyond “do X”

Spec-Driven Development

- Writing specs that are AI-readable and human-readable
- Separating intent from implementation
- Using specs as a stable contract



Vibe Coding

- Co-creating code with AI, not over-planning
- Letting AI “fill in the blanks” responsibly

Agentic Workflows

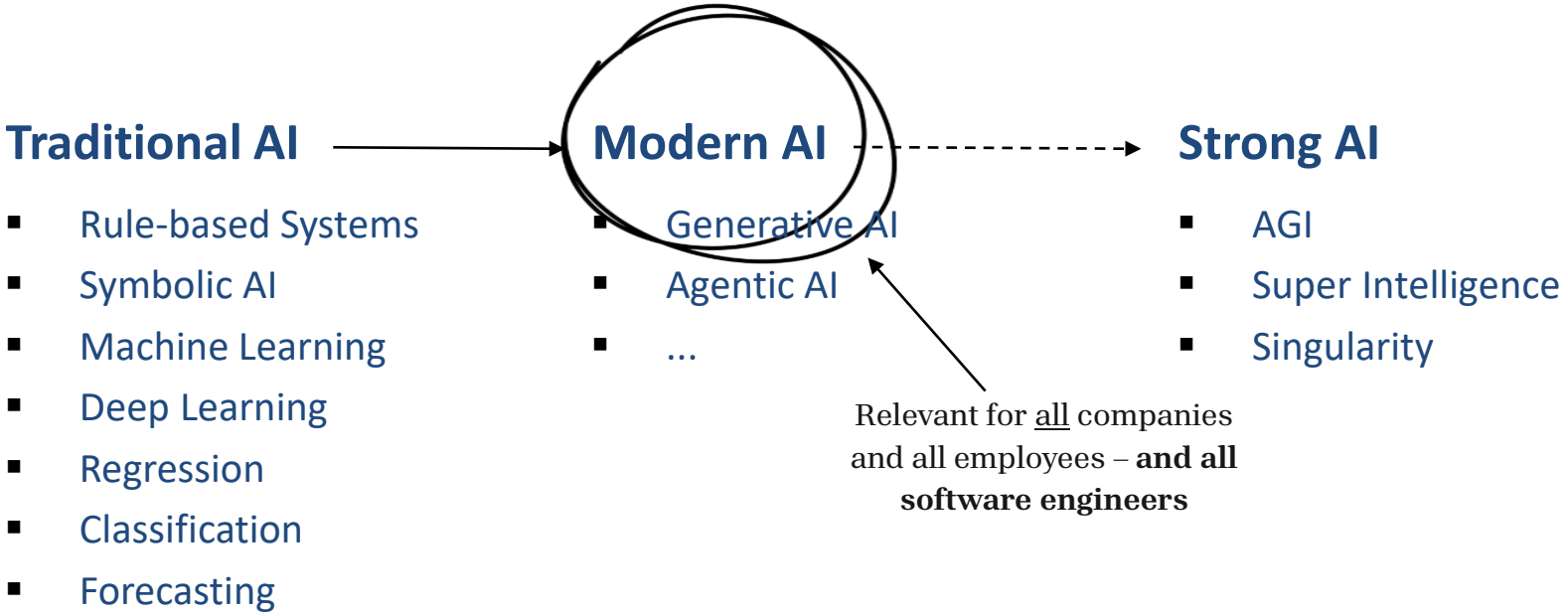
- Agent primitives (plan, act, observe, reflect)
- Context engineering (what the AI needs to know)
- Multi-step, goal-oriented workflows

Agent Framework

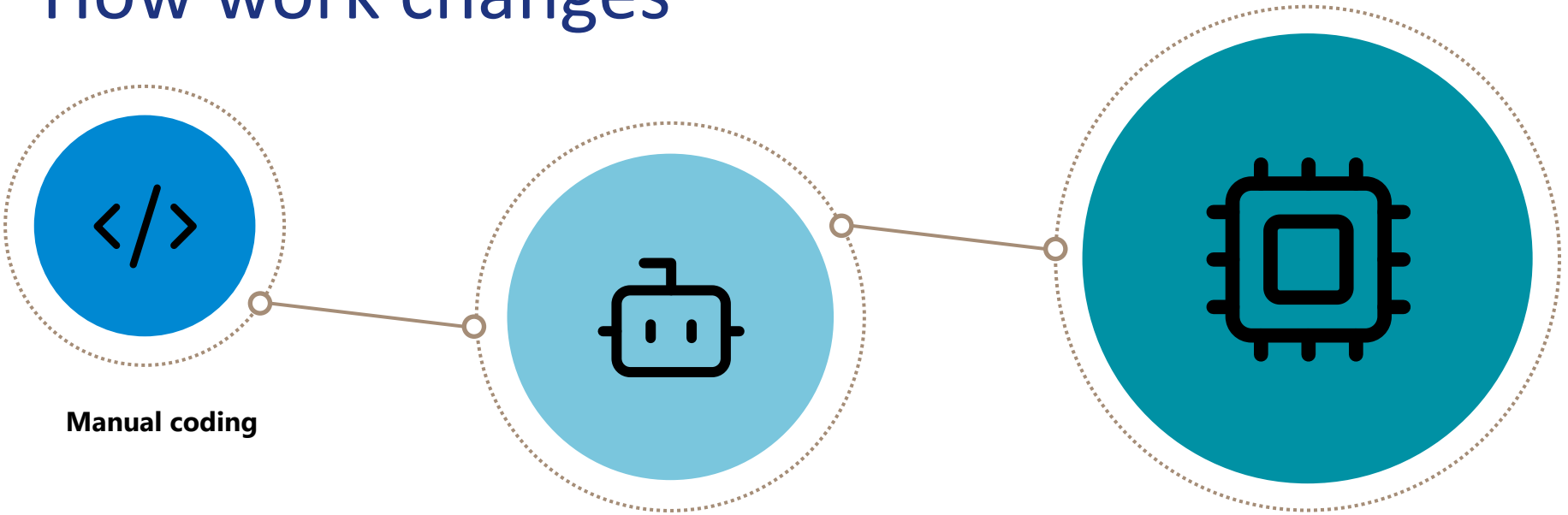
- Agent framework concepts
- Tooling and orchestration
- Testing AI-driven systems
- Validating code and agent behavior

Introduction to modern software development

Modern AI


Traditional AI
Rule-based Systems
Symbolic AI
Machine Learning
Deep Learning
Regression
Classification
Forecasting
Modern AI
Generative AI
Agentic AI
...
Strong AI
AGI
Super Intelligence
Singularity
Relevant for all companies and all employees – and all software engineers

How work changes



Manual coding

AI-assisted development

AI-orchestrated systems

- Developer writes everything
- Manual scaffolding
- Hand-written tests
- Large code surface

Producer → Intent designer + reviewer

- Developer defines intent
- Spec-driven planning
- Agent execution
- Human validation gates

CASE

Getting started

Agenda

Part 1
AI Tools and IDEs
AI Engines and Models
Vibe Coding / prototype
75m

Part 2
AI-Assisted Development
Agentic Workflows
75m

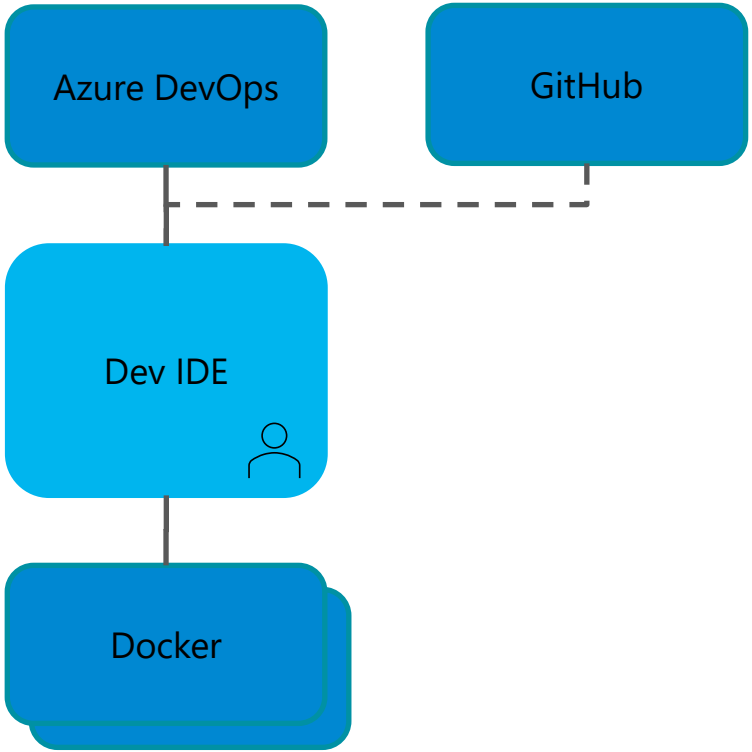
Part 3
Spec-Driven Development
Agent Driven Development
75m

Part 4
Microsoft Agent Framework
AI for automated testing
75m

Part 5
Wrap up
30m

AI Tools and IDEs
AI Engines and Models

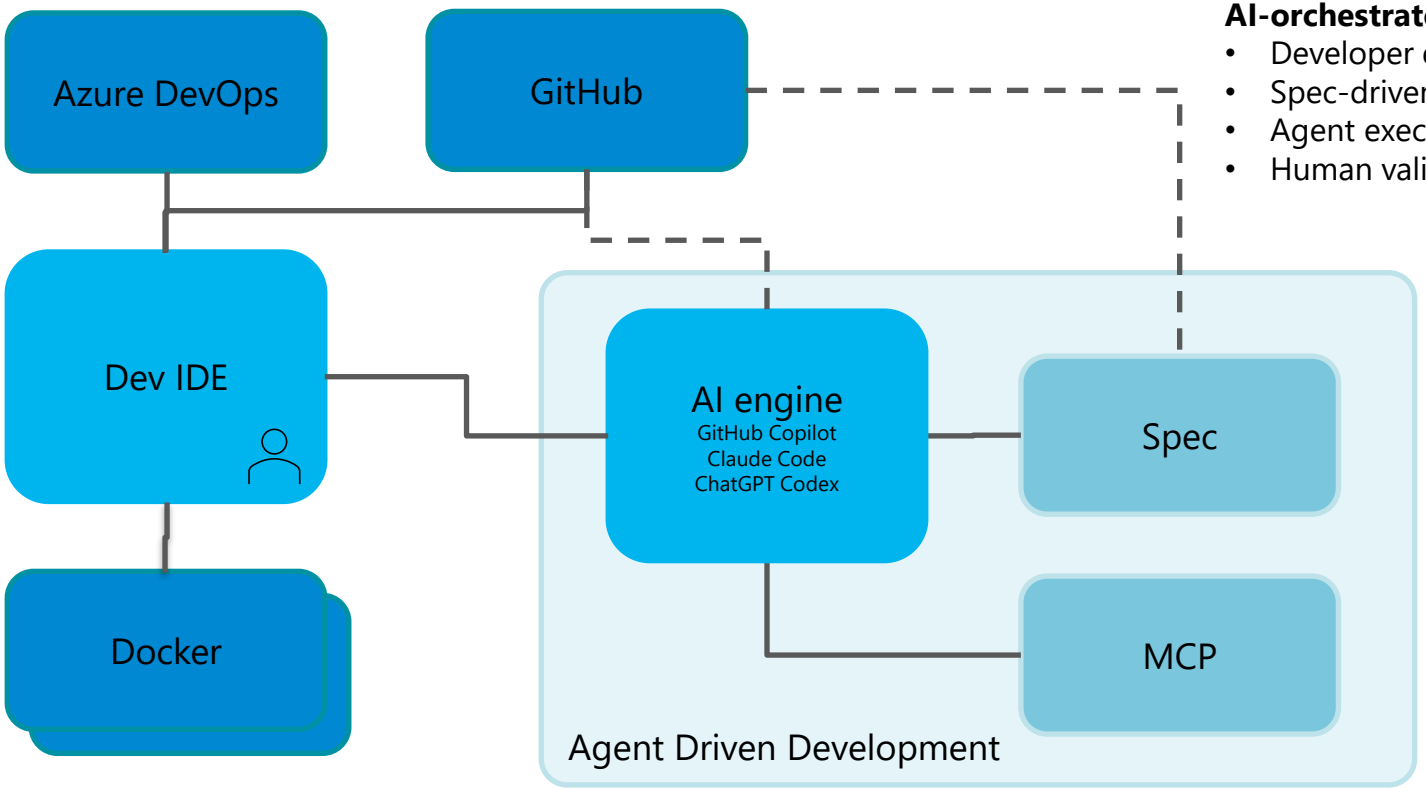
Manual coding setup



Manual coding

- Developer writes everything
- Manual scaffolding
- Hand-written tests
- Large code surface

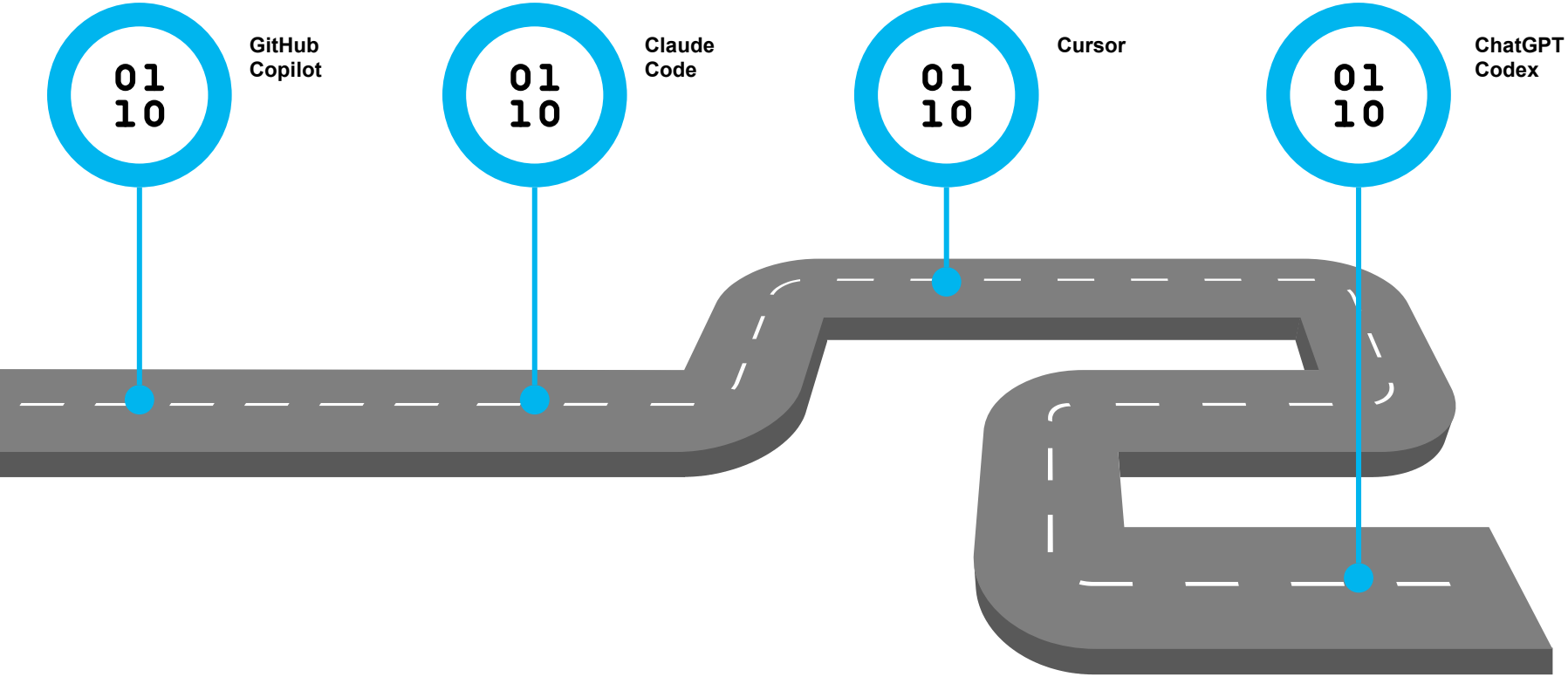
AI-orchestration setup



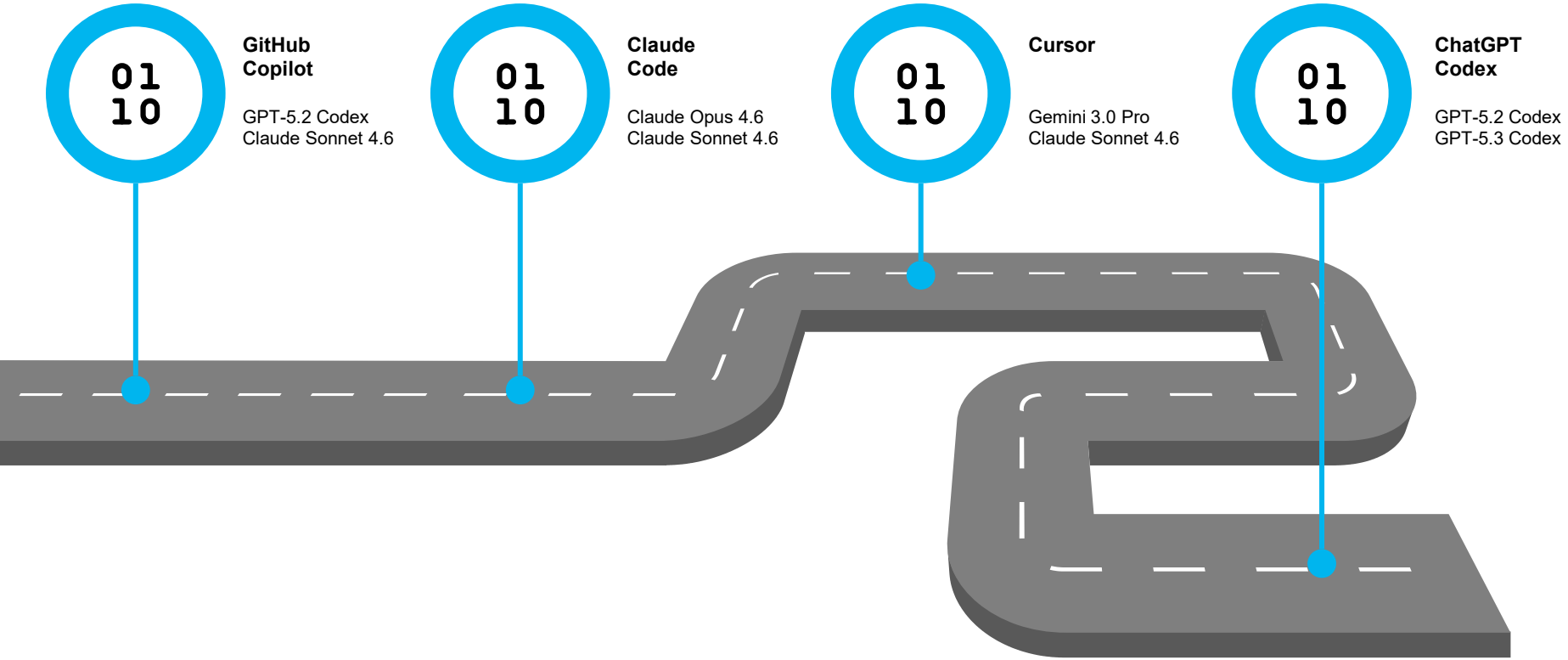
AI-orchestrated systems

- Developer defines intent
- Spec-driven planning
- Agent execution
- Human validation gates

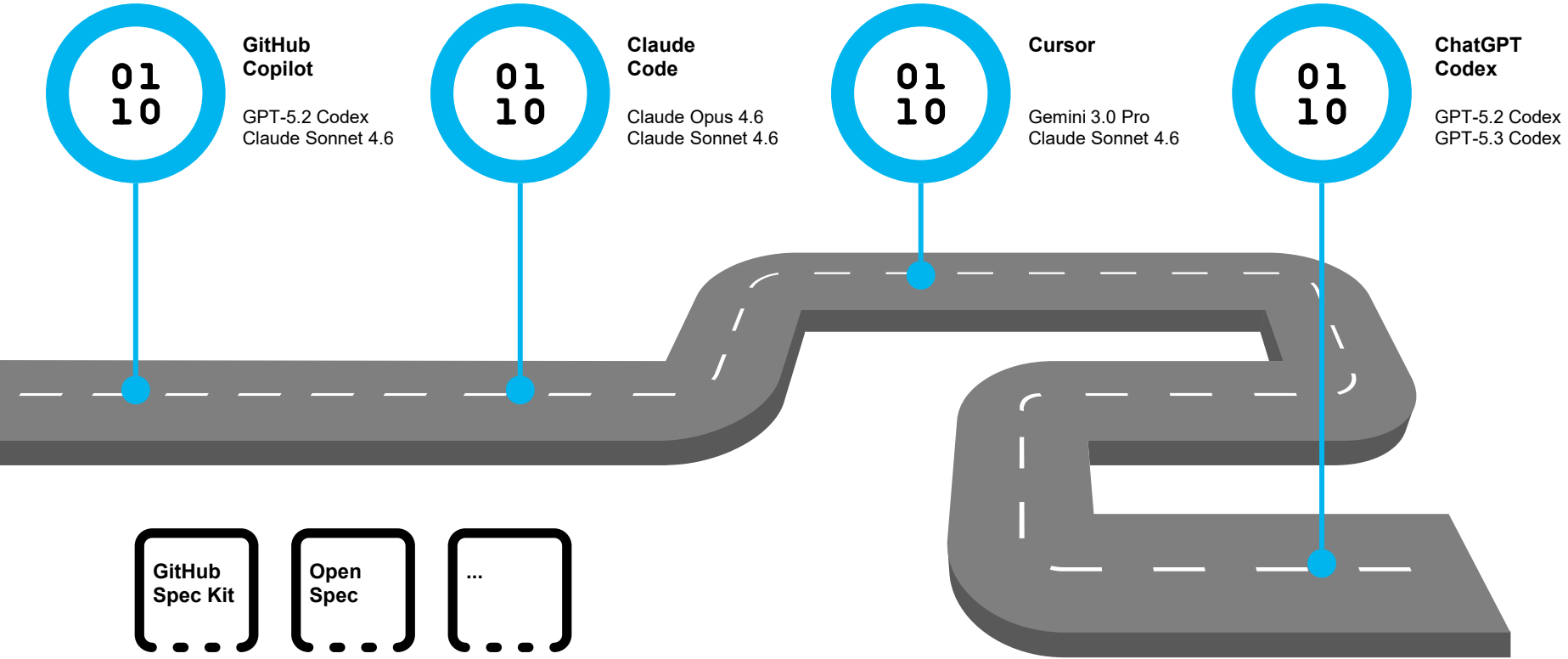
Know your AI tools



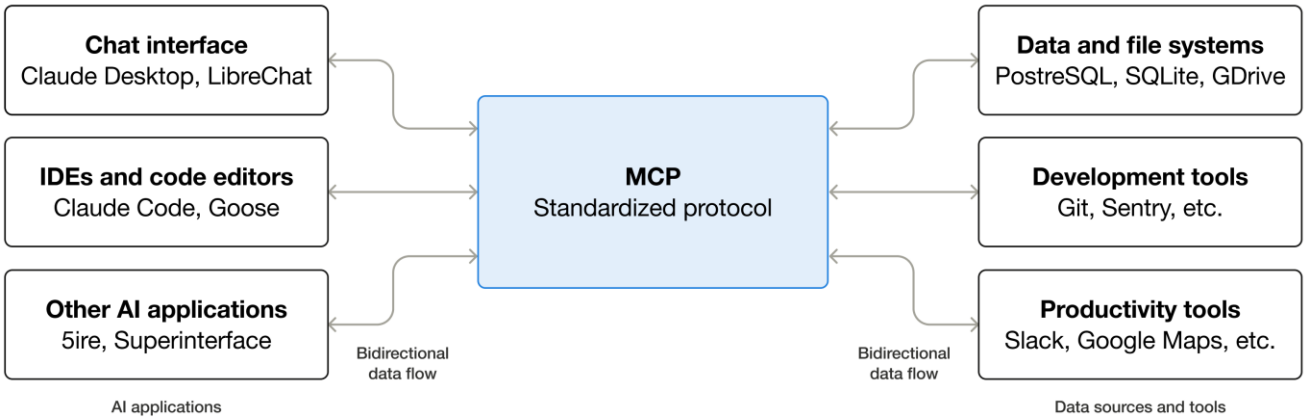
Know your AI tools and your models



Know your AI tools and your models



MCP



https://modelcontextprotocol.io/

DEMO

AI tools and models in action

GitHub Copilot + ChatGPT Codex + OpenCode + OpenSpec + MCP

CASE

AI tools and models

AI Tools and Models

AI Tools and Models

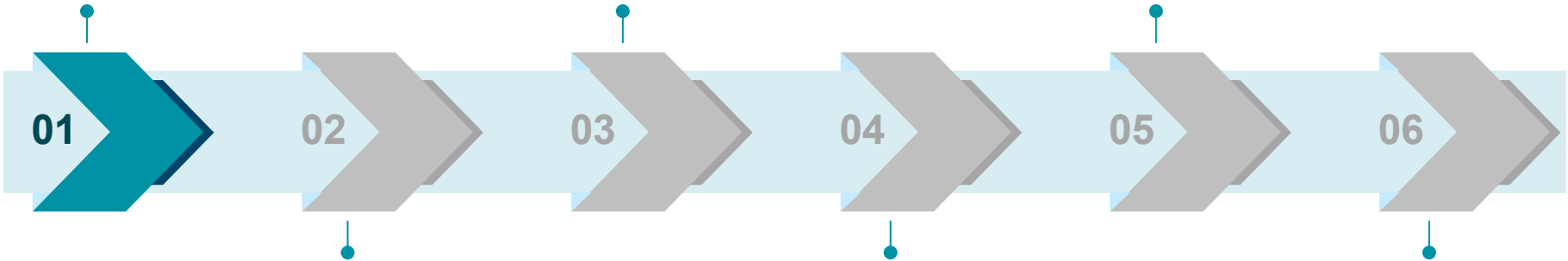
- Using AI tools and IDE integrations effectively
- Understanding model strengths and limitations

AI-Assisted Development

- Prompt engineering beyond "do X"

Spec-Driven Development

- Writing specs that are AI-readable and human-readable
- Separating intent from implementation
- Using specs as a stable contract



Vibe Coding

- Co-creating code with AI, not over-planning
- Letting AI "fill in the blanks" responsibly

Agentic Workflows

- Agent primitives (plan, act, observe, reflect)
- Context engineering (what the AI needs to know)
- Multi-step, goal-oriented workflows

Agent Framework

- Agent framework concepts
- Tooling and orchestration
- Testing AI-driven systems
- Validating code and agent behavior

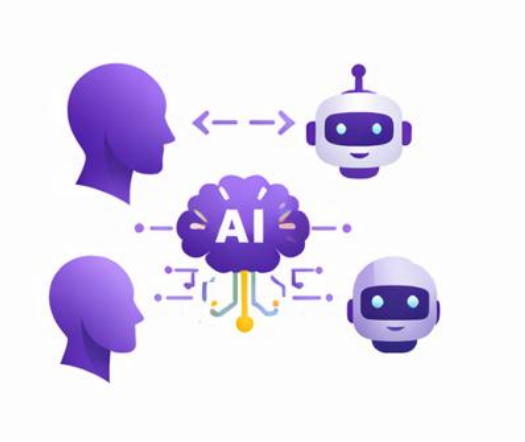
Vibe Coding

Why Vibe Coding exists

- Software complexity keeps increasing
- Software systems are growing in complexity
- AI is good at repetition and wiring
- Developers spend significant time on mechanical work

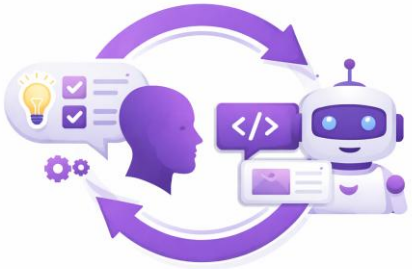
Human
Intent
Judgment

AI
Generation
Iteration



Vibe coding shifts focus from typing code to describing outcomes

What is Vibe Coding?



- Intent-driven development
- Conversational, iterative process
- Output quality depends on human input

Vibe Coding = describing what you want,
how it should feel and what it should look like

"The quality of the result depends on the quality of the direction"

Human → AI → Output → Review

DEMO

Vibe Coding with GitHub Spark

Vibe Coding Platforms Today

Level 1 – Inside the IDE

- Cursor
- Windsurf

Level 2 – Solution-Level builders

- GitHub Spark

Level 3 – Product-Level Generators

- Figma Make
- Lovable
- Emergent

Same mindset – different abstraction levels

What Vibe Coding is – and is not

It is:
- Fast feedback
- Great for exploration
- A new way to collaborate
- Accelerating prototype and MVP solutions

It is not:
- Thinking-free
- Guaranteed clean architecture
- Free of responsibility

"Because something is simple – doesn't necessarily make it easy"

CASE

Vibe Coding

https://github.com/ELDK26-AI-masterclass/case

Vibe Coding

AI Tools and Models

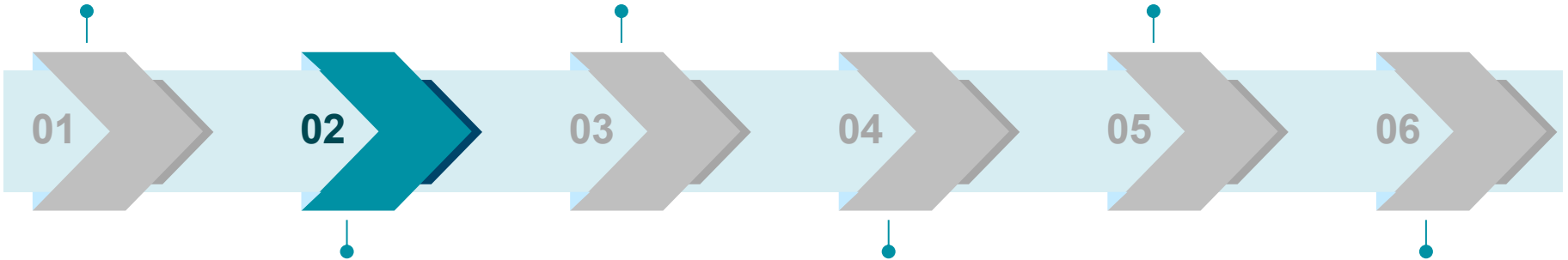
- Using AI tools and IDE integrations effectively
- Understanding model strengths and limitations

AI-Assisted Development

- Prompt engineering beyond “do X”

Spec-Driven Development

- Writing specs that are AI-readable and human-readable
- Separating intent from implementation
- Using specs as a stable contract



Vibe Coding

- Co-creating code with AI, not over-planning
- Letting AI “fill in the blanks” responsibly

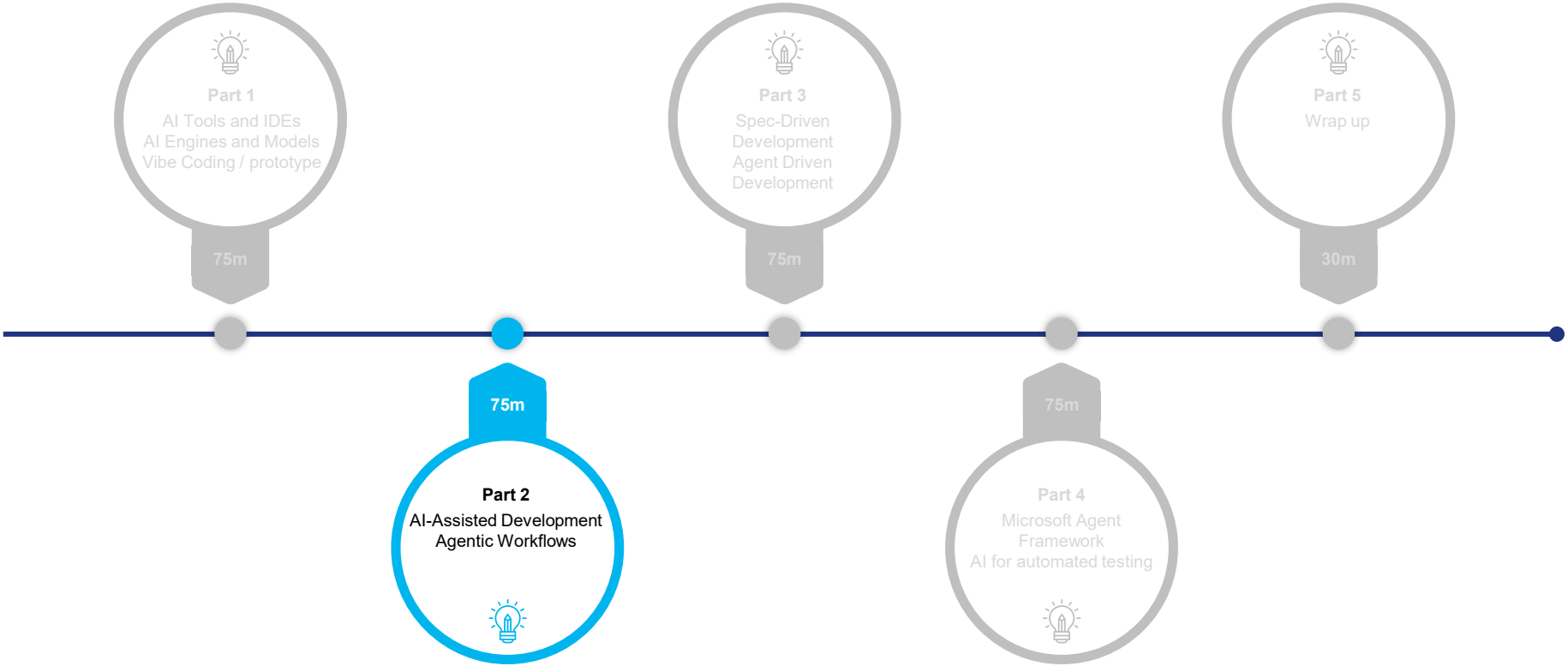
Agentic Workflows

- Agent primitives (plan, act, observe, reflect)
- Context engineering (what the AI needs to know)
- Multi-step, goal-oriented workflows

Agent Framework

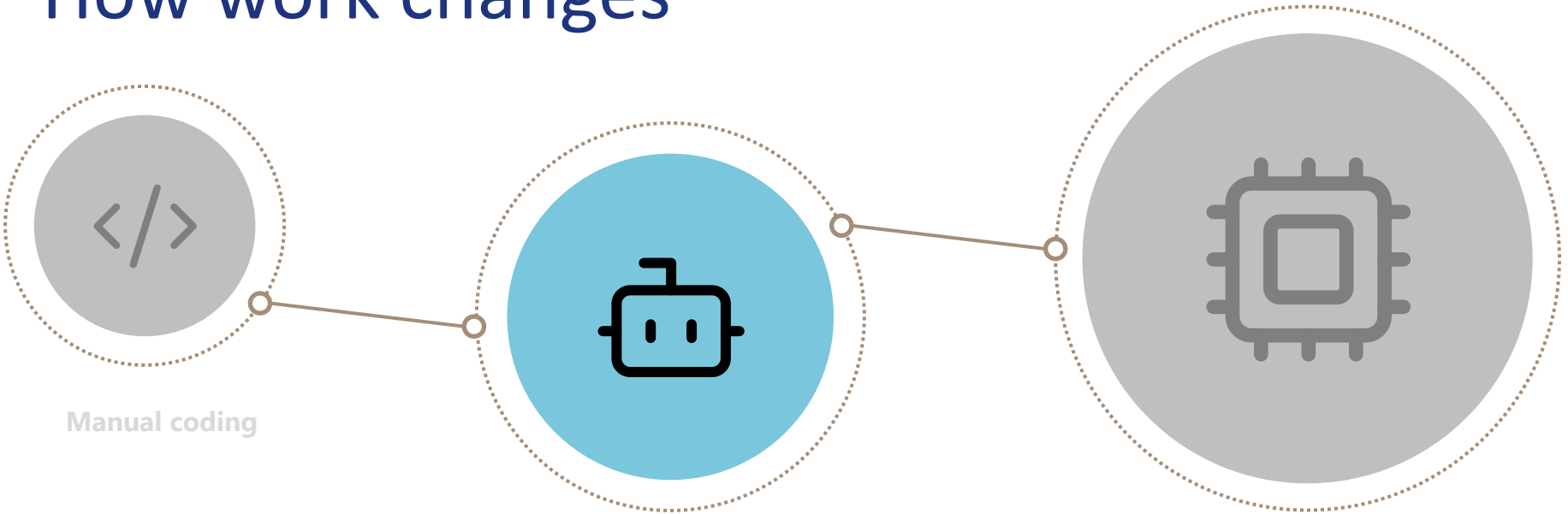
- Agent framework concepts
- Tooling and orchestration
- Testing AI-driven systems
- Validating code and agent behavior

Agenda


Part 1
AI Tools and IDEs
AI Engines and Models
Vibe Coding / prototype
75m
Part 3
Spec-Driven Development
Agent Driven Development
75m
Part 5
Wrap up
30m
75m
Part 2
AI-Assisted Development
Agentic Workflows
75m
Part 4
Microsoft Agent Framework
AI for automated testing

AI-assisted development

How work changes

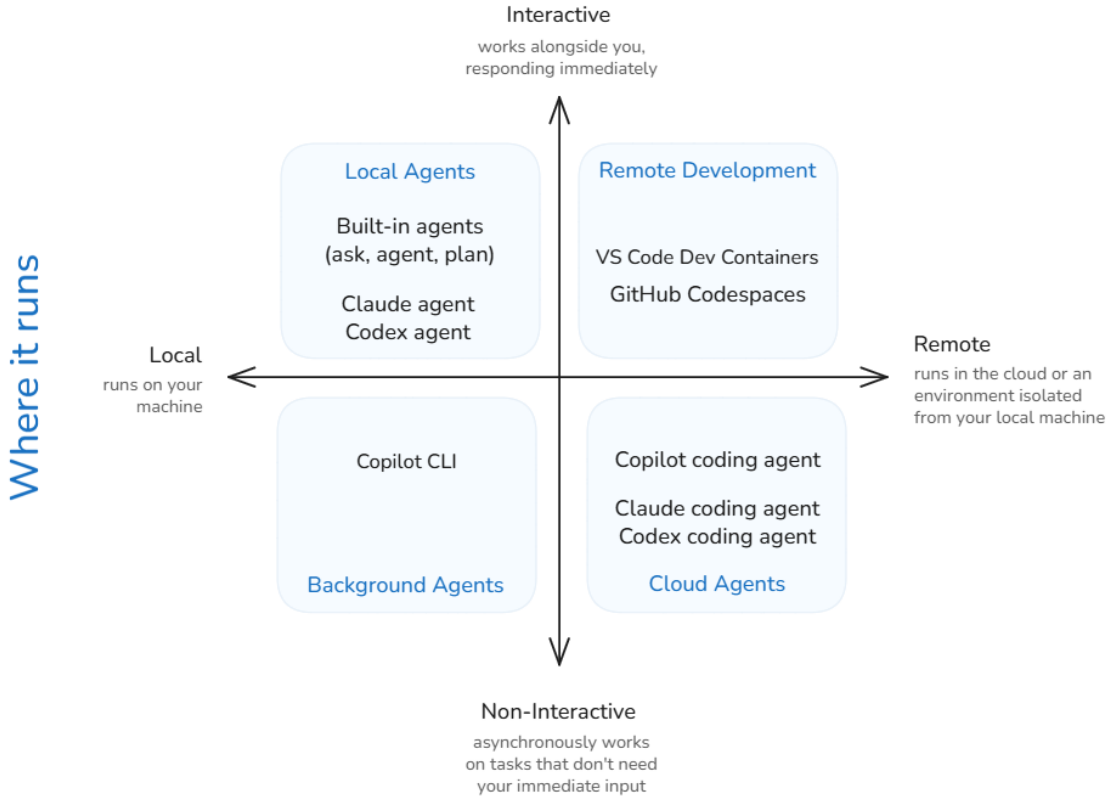


- Developer writes everything
- Manual scaffolding
- Hand-written tests
- Large code surface

Producer → Intent designer + reviewer

- Developer defines intent
- Spec-driven planning
- Agent execution
- Human validation gates

Agent Sessions



DEMO

Agent Sessions

Local + Remote

How agents run

Local

Local execution in the IDE

- Best when: early iteration
- Benefits: fast feedback
- Example: building first .spec.md

Async

Async delegation

- Best when: repetitive tasks
- Benefits: runs in background
- Example: test generation

Hybrid

Hybrid orchestration

- Best when: multi-agent flows
- Benefits: scalable implementation
- Example: feature spec → tests → docs

CASE

AI-assisted development

AI-Assisted Development

AI Tools and Models

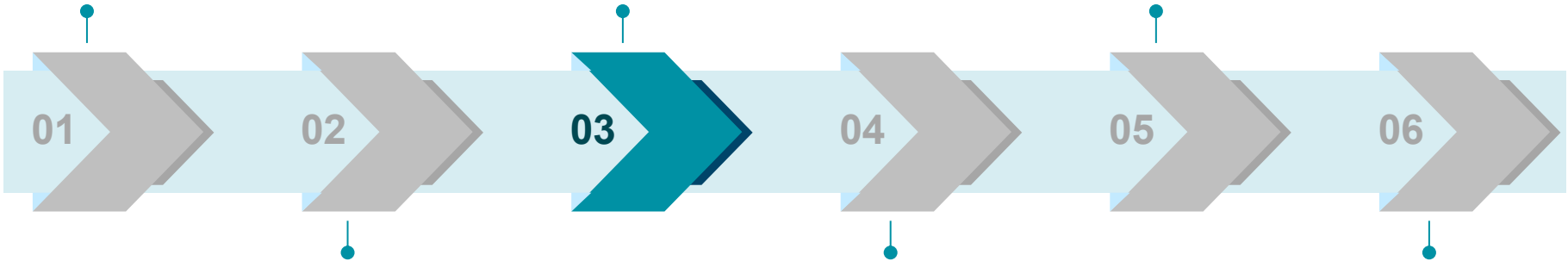
- Using AI tools and IDE integrations effectively
- Understanding model strengths and limitations

AI-Assisted Development

- Prompt engineering beyond “do X”

Spec-Driven Development

- Writing specs that are AI-readable and human-readable
- Separating intent from implementation
- Using specs as a stable contract



Vibe Coding

- Co-creating code with AI, not over-planning
- Letting AI “fill in the blanks” responsibly

Agentic Workflows

- Agent primitives (plan, act, observe, reflect)
- Context engineering (what the AI needs to know)
- Multi-step, goal-oriented workflows

Agent Framework

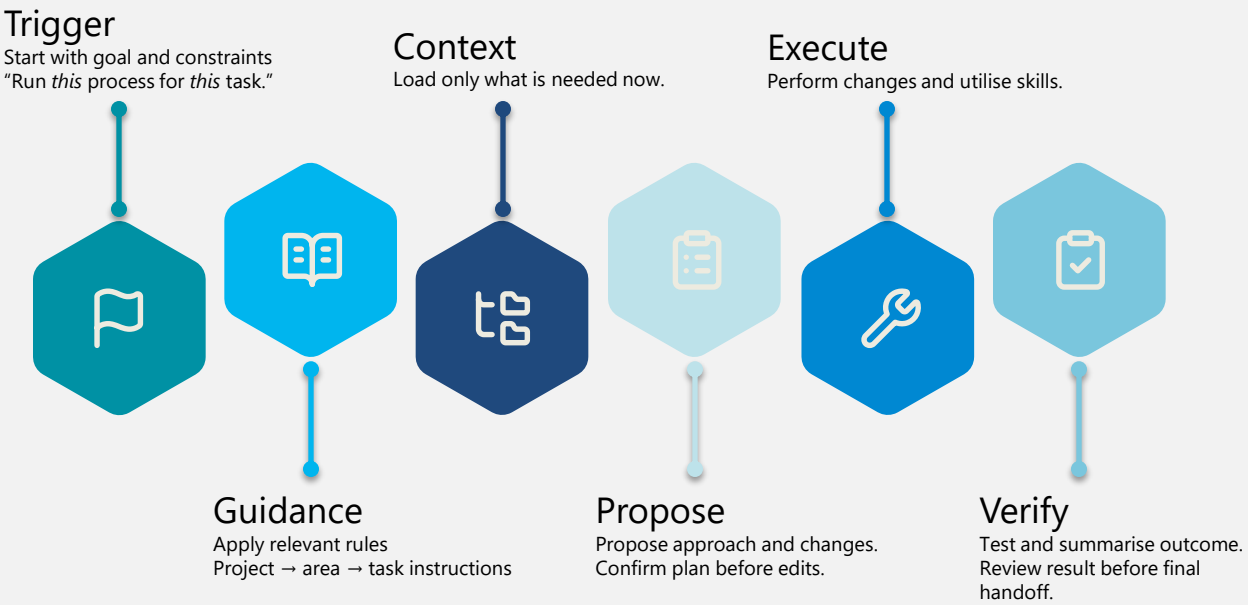
- Agent framework concepts
- Tooling and orchestration
- Testing AI-driven systems
- Validating code and agent behavior

Agentic Workflows

Agentic Workflows

Agentic workflow is a reusable process encoded as a prompt.

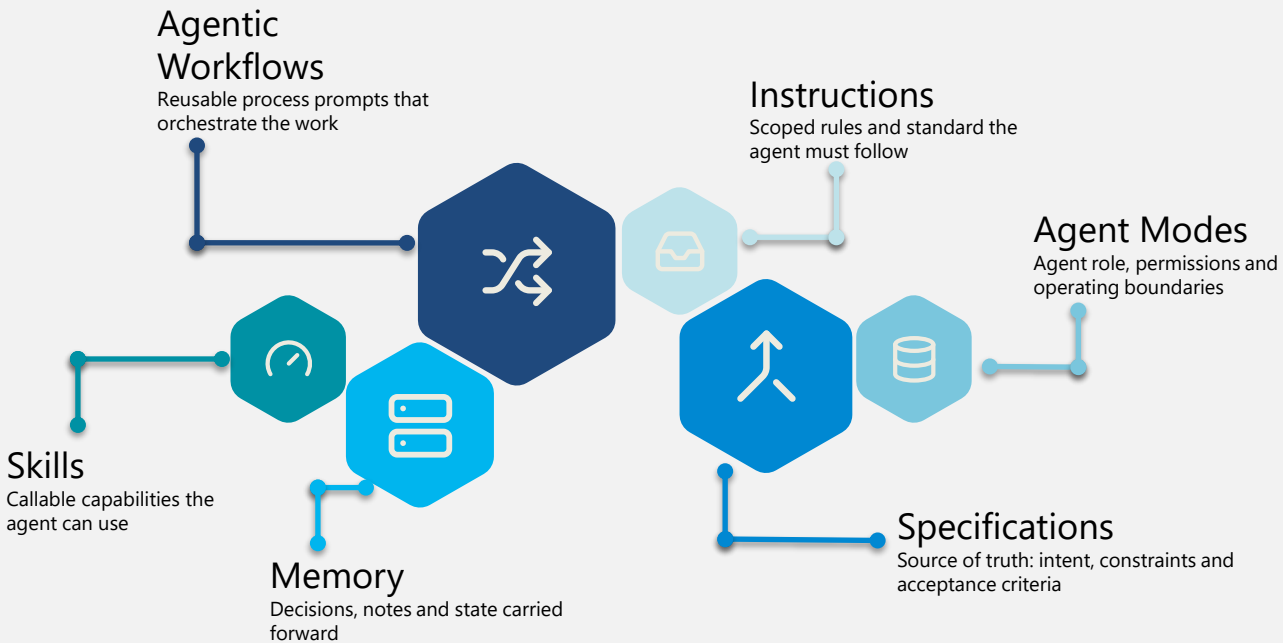
- Captured in a workflow prompt
- Orchestrates Agent Primitives
- Runs in phases
- Goal is consistent outcomes



Agentic Building Blocks

Agentic workflows scale when we externalize behavior into project assets.

Agentic Building Blocks make work repeatable, reviewable and shared across the team.



DEMO

Agentic workflows

Instructions + Specifications + Skills

CASE

Agentic workflows

Agentic Workflows

AI Tools and Models

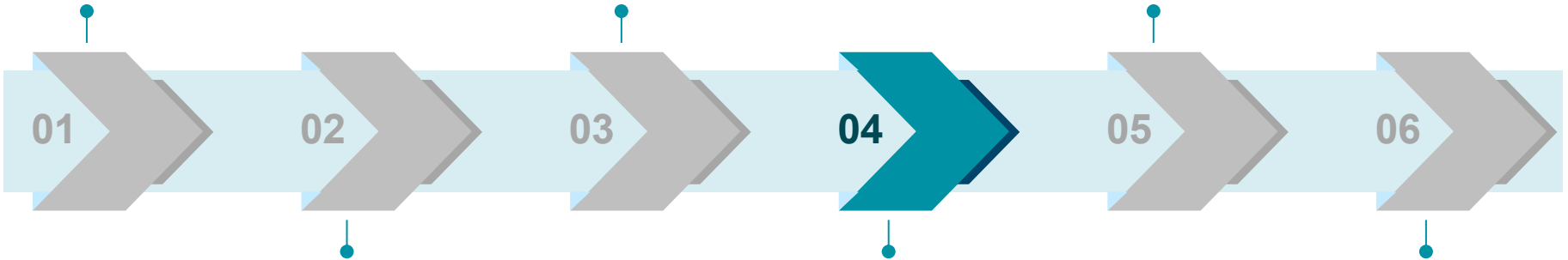
- Using AI tools and IDE integrations effectively
- Understanding model strengths and limitations

AI-Assisted Development

- Prompt engineering beyond “do X”

Spec-Driven Development

- Writing specs that are AI-readable and human-readable
- Separating intent from implementation
- Using specs as a stable contract



Vibe Coding

- Co-creating code with AI, not over-planning
- Letting AI “fill in the blanks” responsibly

Agentic Workflows

- Agent primitives (plan, act, observe, reflect)
- Context engineering (what the AI needs to know)
- Multi-step, goal-oriented workflows

Agent Framework

- Agent framework concepts
- Tooling and orchestration
- Testing AI-driven systems
- Validating code and agent behavior

Agenda

Part 1
AI Tools and IDEs
AI Engines and Models
Vibe Coding / prototype
75m

Part 2
AI-Assisted Development
Agentic Workflows
75m

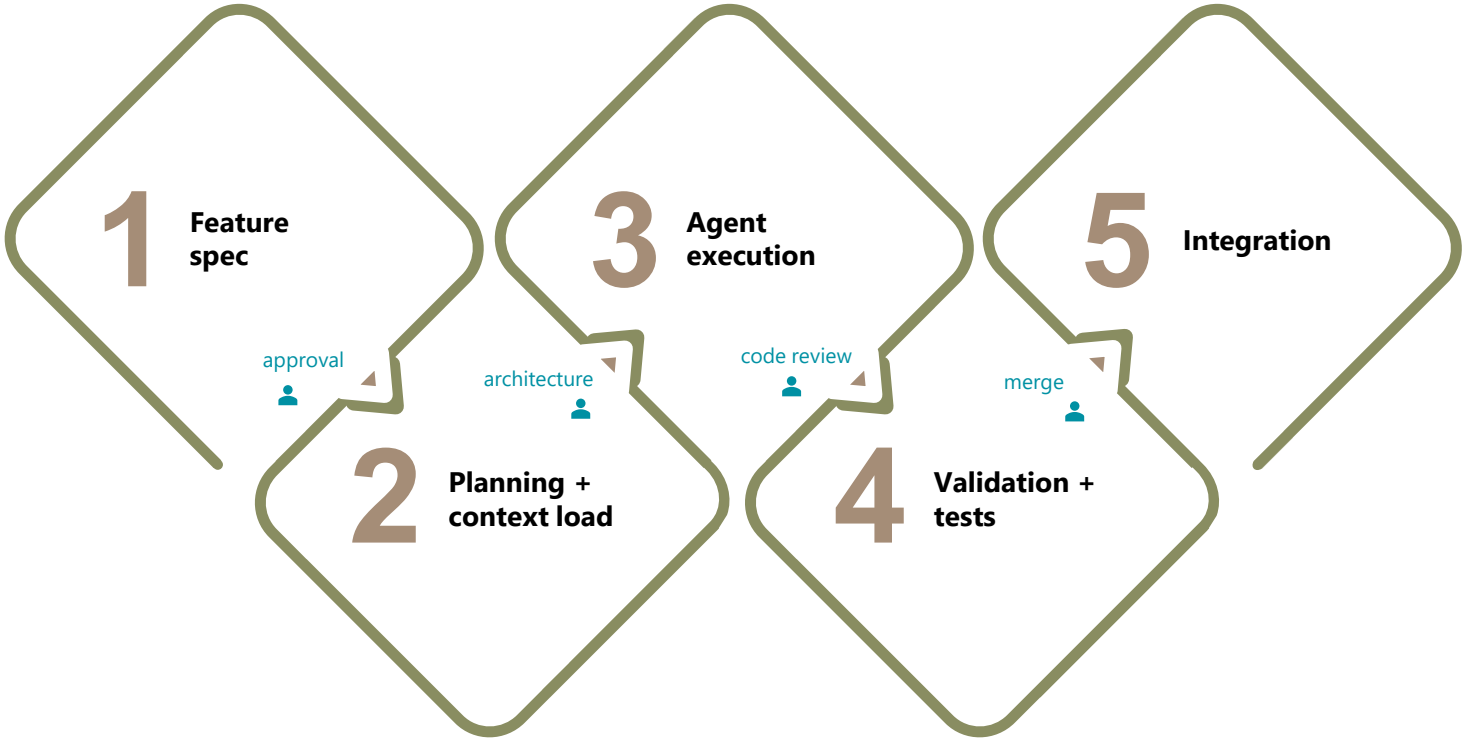
Part 3
Spec-Driven Development
Agent Driven Development
75m

Part 4
Microsoft Agent Framework
AI for automated testing
75m

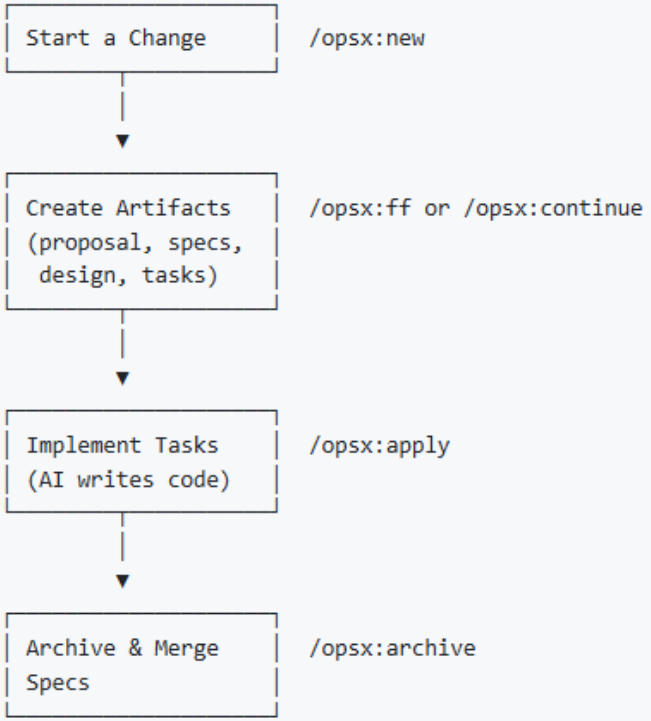
Part 5
Wrap up
30m

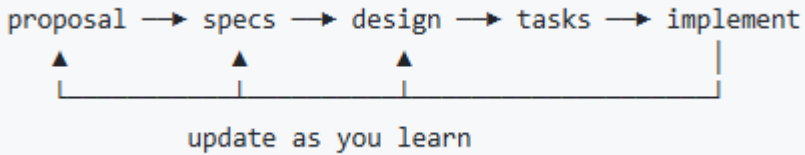
Spec-Driven Development

Spec-driven development



OpenSpec

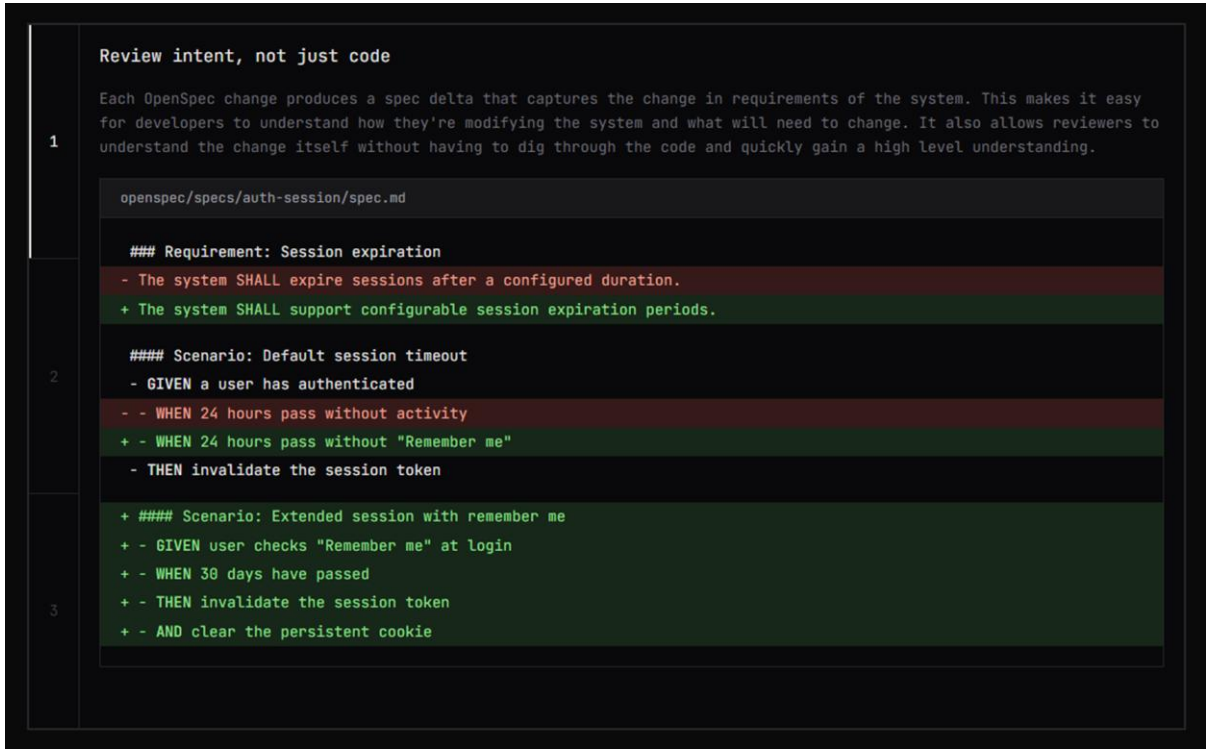

Start a Change
/opsx:new
Create Artifacts (proposal, specs, design, tasks)
/opsx:ff or /opsx:continue
Implement Tasks (AI writes code)
/opsx:apply
Archive & Merge Specs
/opsx:archive


proposal
specs
design
tasks
implement
update as you learn

Change – Implement – Archive

```
You: /opsx:new add-dark-mode
AI:  Created openspec/changes/add-dark-mode/
     Ready to create: proposal

You: /opsx:ff   # "fast-forward" - generate all planning docs
AI:  ✓ proposal.md – why we're doing this, what's changing
     ✓ specs/       – requirements and scenarios
     ✓ design.md    – technical approach
     ✓ tasks.md     – implementation checklist
     Ready for implementation!

You: /opsx:apply
AI:  Implementing tasks...
     ✓ 1.1 Add theme context provider
     ✓ 1.2 Create toggle component
     ✓ 2.1 Add CSS variables
     ✓ 2.2 Wire up localStorage
     All tasks complete!

You: /opsx:archive
AI:  Archived to openspec/changes/archive/2025-01-23-add-dark-mode/
     Specs updated. Ready for the next feature.
```

Review changes

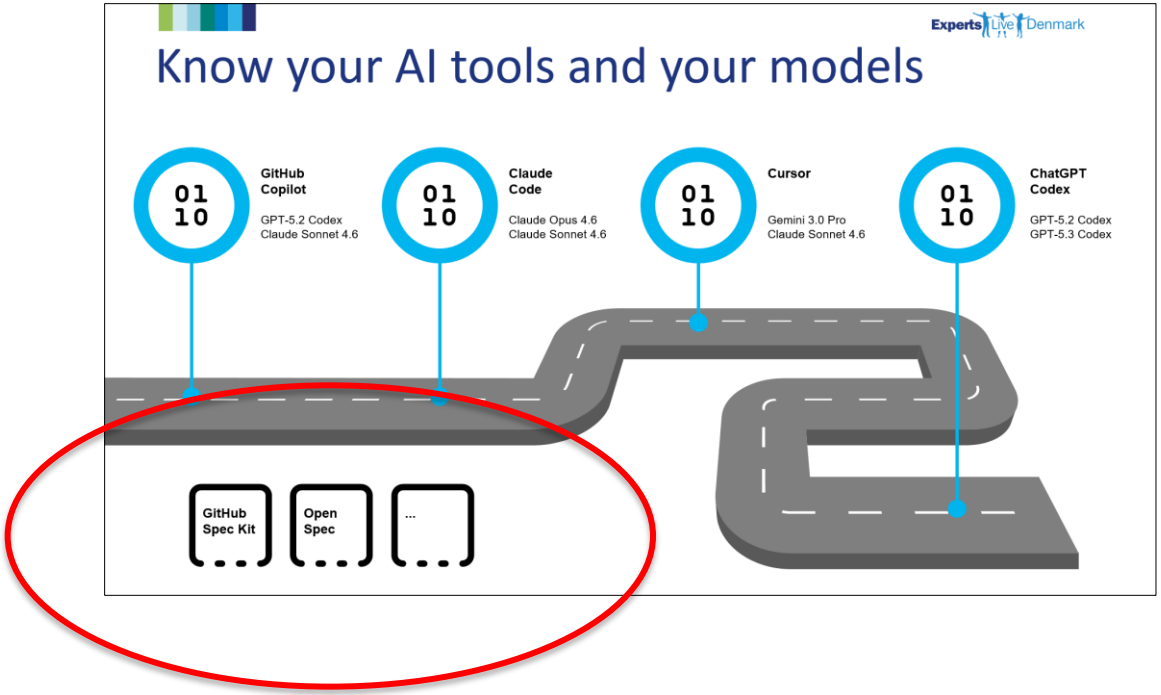


DEMO

Spec-driven development

OpenSpec: change – implement – archive

Spec Engines

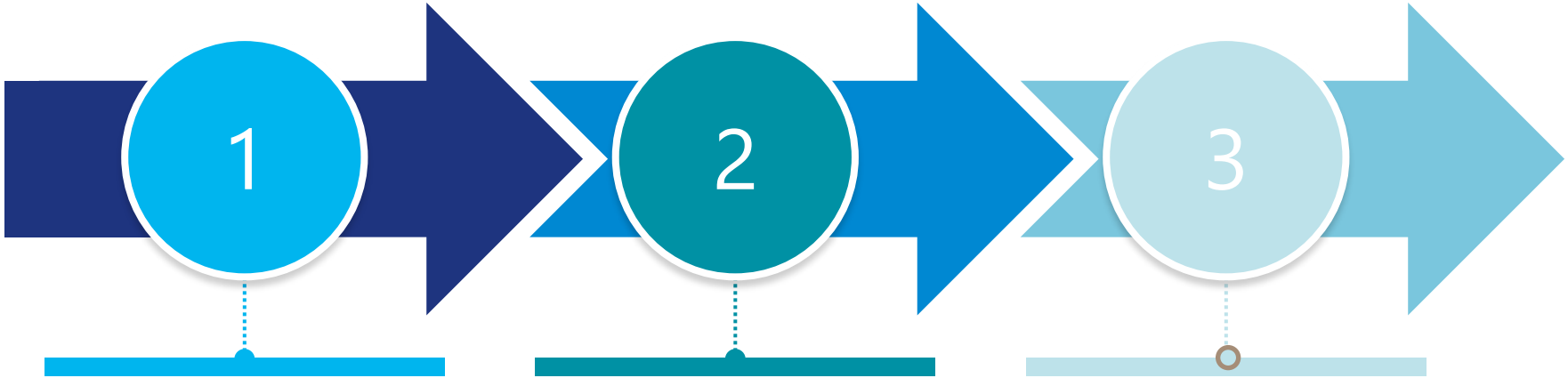


The change into spec-driven development

Requirements → Code	❯	Specification → Code → Validation
Documentation optional	❯	Specification is required
Tests define behavior	❯	Specification defines purpose and intent
AI guesses intent	❯	AI follows intent

Agent-Driven Development

Agent Driven Development



KNOWLEDGE

- Description of customer, industry, and previous projects
- Description of infrastructure, governance, and compliance
- Description of project, audience, roles, and security
- Description of tech stack, architecture, and AI capabilities

PRIMITIVES

- Import of customer, Twoday and third-party primitives
- Generation of project specific primitives including agents, instructions, chat modes, and prompts
- Compilation of primitives into standard primitives format

ORCHESTRATION

- Use spec engine to work on features and tasks
- Follow spec-driven principals: Change => Implement => Archive => Close
- Use human orchestration and review, and AI test and review

DEMO

Agent-driven development

Putting the pieces together

CASE

Spec-driven development

Putting the pieces together

Spec-Driven Development

AI Tools and Models

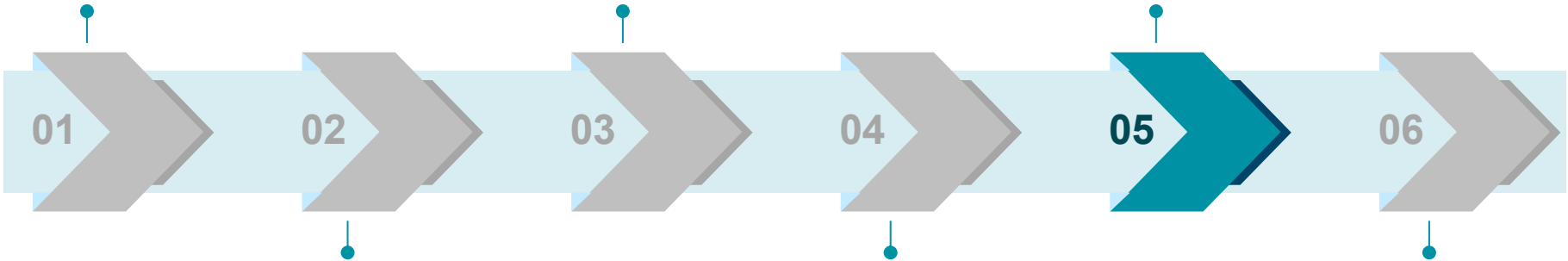
- Using AI tools and IDE integrations effectively
- Understanding model strengths and limitations

AI-Assisted Development

- Prompt engineering beyond "do X"

Spec-Driven Development

- Writing specs that are AI-readable and human-readable
- Separating intent from implementation
- Using specs as a stable contract



Vibe Coding

- Co-creating code with AI, not over-planning
- Letting AI "fill in the blanks" responsibly

Agentic Workflows

- Agent primitives (plan, act, observe, reflect)
- Context engineering (what the AI needs to know)
- Multi-step, goal-oriented workflows

Agent Framework

- Agent framework concepts
- Tooling and orchestration
- Testing AI-driven systems
- Validating code and agent behavior

Agenda

Part 1
AI Tools and IDEs
AI Engines and Models
Vibe Coding / prototype
75m

Part 2
AI-Assisted Development
Agentic Workflows
75m

Part 3
Spec-Driven Development
Agent Driven Development
75m

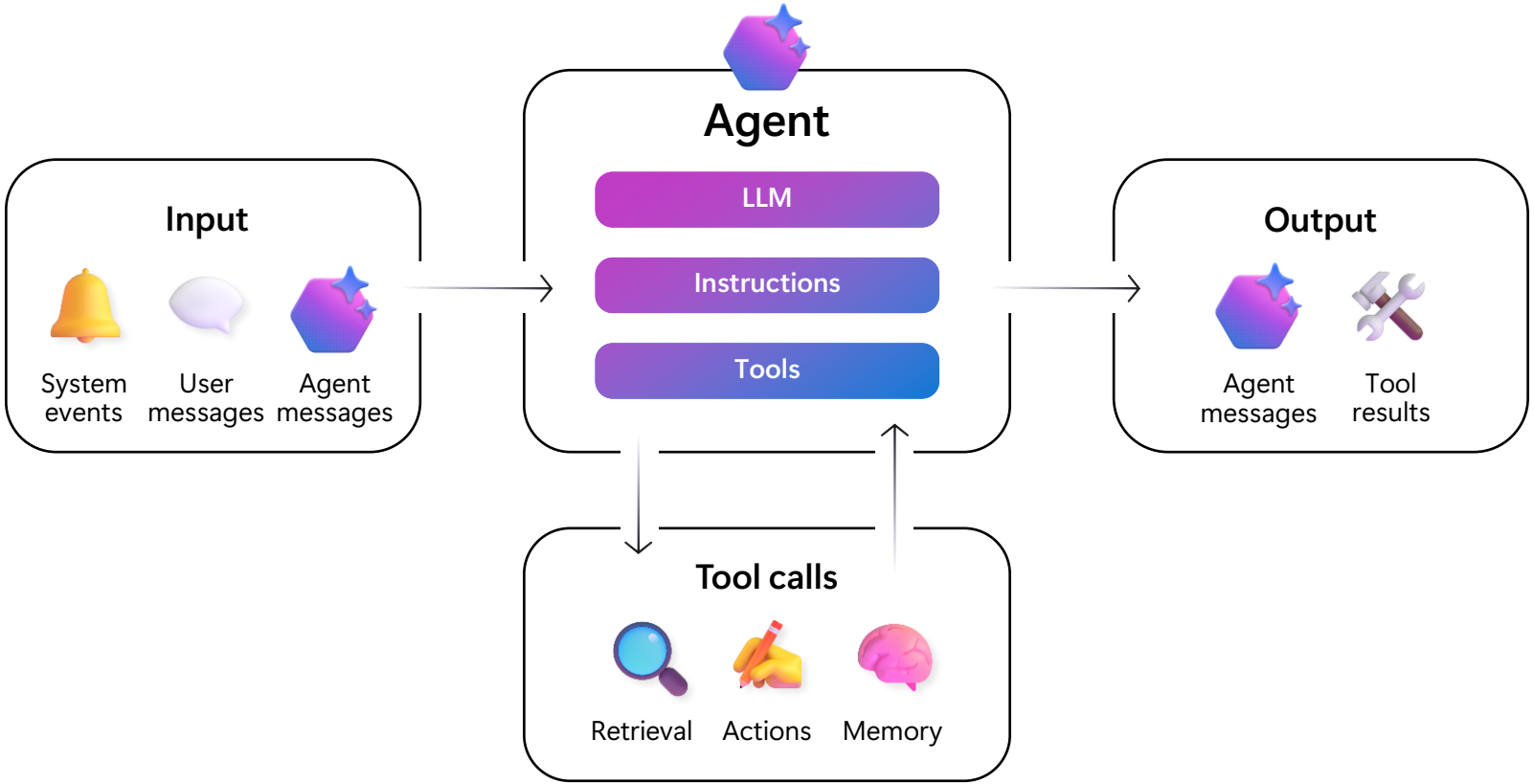
Part 4
Microsoft Agent Framework
AI for automated testing
75m

Part 5
Wrap up
30m

Part 4

Developing your own agents for AI-assisted development

What is an agent?



Part 4 Agenda

In the last part of this master class, we're moving on from using existing AI tools to building our own!

Plan for the rest of this part:

- Introduction of the tools and frameworks (15 min)
- Hands-on exercise (60 min)

Experts Live Denmark

To make an AI agent for software development, you'll need…

Serves 1 | Convert to metric

- 1 Large Language Model for the base
- 1 handful of tools (make your own or use MCPs)
- A pinch of prompt engineering (don't skip this)
- A few thousand hours of development time to tie it all together (can be substituted with an existing framework)

Public Preview

Microsoft Agent Framework

Open-source engine for building and orchestrating intelligent AI agents

Open Standards & Interoperability

Pipeline for Research

Community-Driven & Extensible by Design

Ready for Production

aka.ms/AgentFramework

Converging the two popular OSS frameworks

Microsoft Agent Framework

Semantic Kernel

Full SDK designed to build AI agents with ease, excellent for single agents and can be extended for multi-agents with integrations to AutoGen

AutoGen

Powerful multi-agent research framework with pre-built conversation orchestration patterns for handling complex agent systems

Unified SDK & Simple Agent Primitives

Agent Framework introduces unified primitives (`AIAgent`, `AgentThread`, `AgentTool`) built on `Microsoft.Extensions.AI`.

Developers can create and run agents with minimal code and reuse the same abstractions across OpenAI, Foundry SDK, or any LLM backend.

Run Agent

Agents

```
AIAgent agent = new AzureOpenAIClient(
    new Uri(Environment.GetEnvironmentVariable("AZURE_OPENAI_ENDPOINT")!),
    new AzureCliCredential())
     .GetChatClient("gpt-4o-mini")
     .CreateAIAgent(
        instructions: "You are a helpful assistant, you can help the user with weather information.");

Console.WriteLine(await agent.RunAsync("What's the weather in Amsterdam?"));
```

Tools

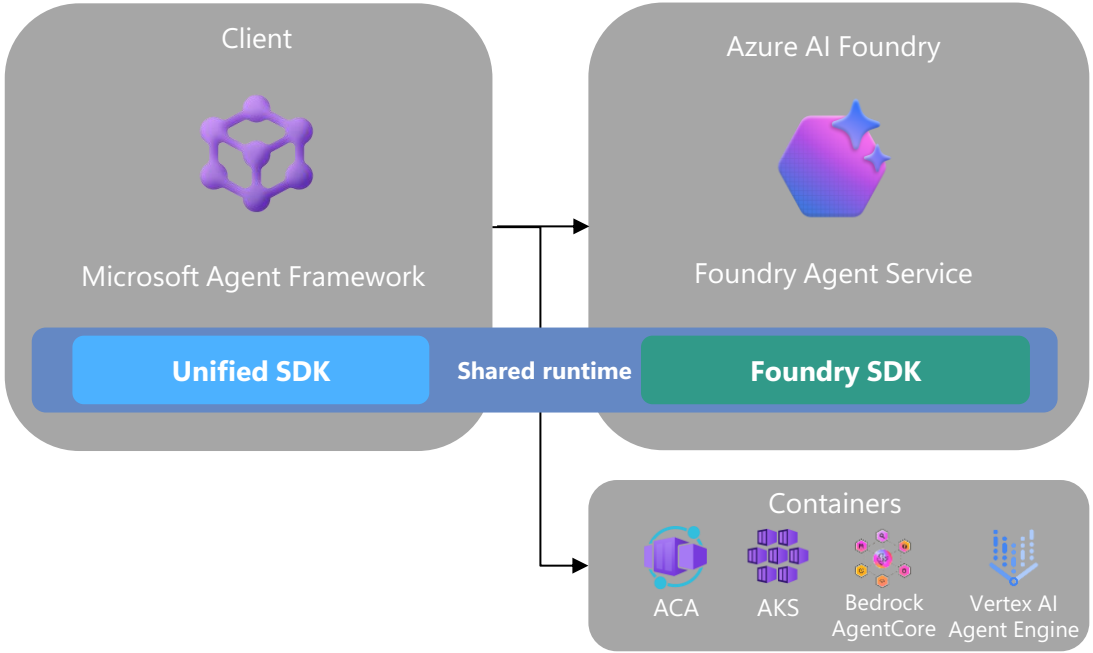
```
[Description("Get the weather for a given location.")]
1 reference | - changes | -authors, -changes
static string GetWeather([Description("The location to get the weather for.")] string location)
    => $"The weather in {location} is cloudy with a high of 15°C.";

AIAgent agent = new AzureOpenAIClient(
    new Uri(Environment.GetEnvironmentVariable("AZURE_OPENAI_ENDPOINT")!),
    new AzureCliCredential())
     .GetChatClient("gpt-4o-mini")
     .CreateAIAgent(
        instructions: "You are a helpful assistant, you can help the user with weather information.",
        tools: [AIFunctionFactory.Create(GetWeather)]);

Console.WriteLine(await agent.RunAsync("What's the weather in Amsterdam?"));
```

Threads

```
[Description("Get the weather for a given location.")]
1 reference | - changes | -authors, -changes
static string GetWeather([Description("The location to get the weather for.")] string location)
    => $"The weather in {location} is cloudy with a high of 15°C.";

AIAgent agent = new AzureOpenAIClient(
    new Uri(Environment.GetEnvironmentVariable("AZURE_OPENAI_ENDPOINT")!),
    new AzureCliCredential())
     .GetChatClient("gpt-4o-mini")
     .CreateAIAgent(
        instructions: "You are a helpful assistant, you can help the user with weather information.",
        tools: [AIFunctionFactory.Create(GetWeather)]);

AgentThread thread = agent.GetNewThread();

Console.WriteLine(await agent.RunAsync("What's the weather in Amsterdam?", thread));

Console.WriteLine(await agent.RunAsync("Is it likely to rain?", thread));
```

Local-First, Cloud-Agnostic

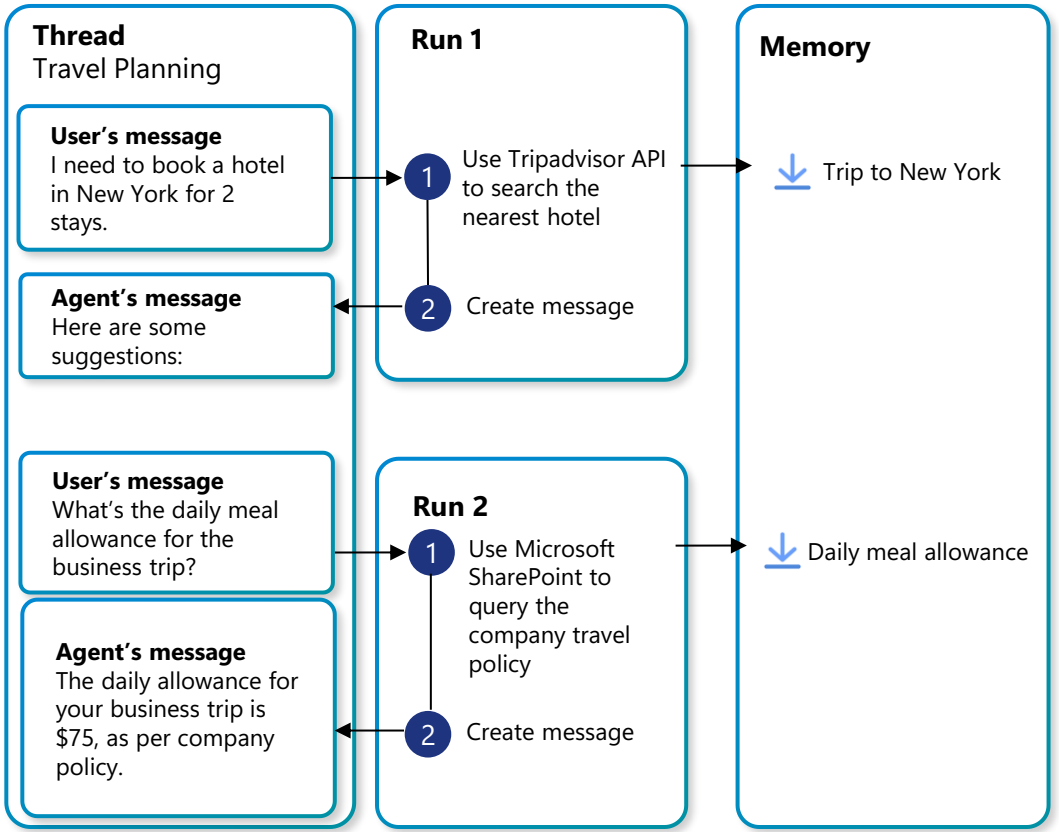
Developers can run agents locally on their laptop, test against their preferred SDK (Azure OpenAI, OpenAI SDK, Foundry SDK), and then move the same code seamlessly to Foundry Agent Service or any cloud containers without rewrites.



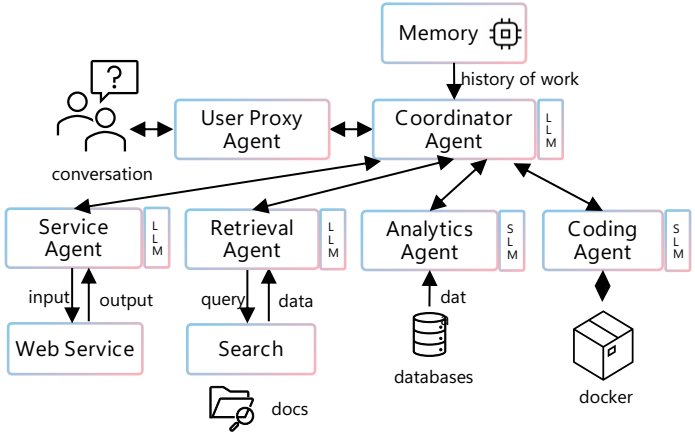
Memory & Conversation State

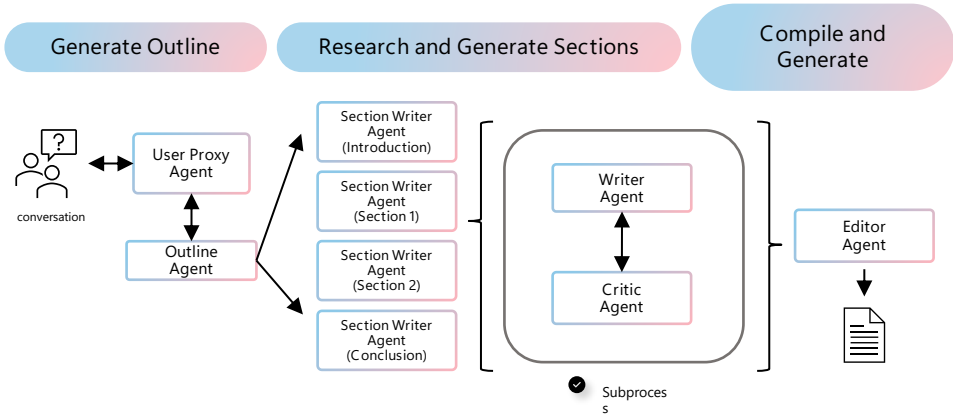
The AgentThread abstraction retains conversation history across turns and sessions, ensuring agents have relevant context for long-running dialogues.

Agent Framework provides **short-term memory** (session-scoped, immediate context) and **long-term memory** (persistent data for user preferences, facts, etc.) through integrations with vector DBs.



Agent systems with Agent Framework





Agent Orchestration

LLM-driven orchestration

Creative reasoning and decision-making

Ex: Propose 2 Instagram marketing campaigns including assets that would leverage the top 2 recent trends in our past quarter US Sales to boost our mailing list user base and predict the impact of each campaign

Workflow Orchestration

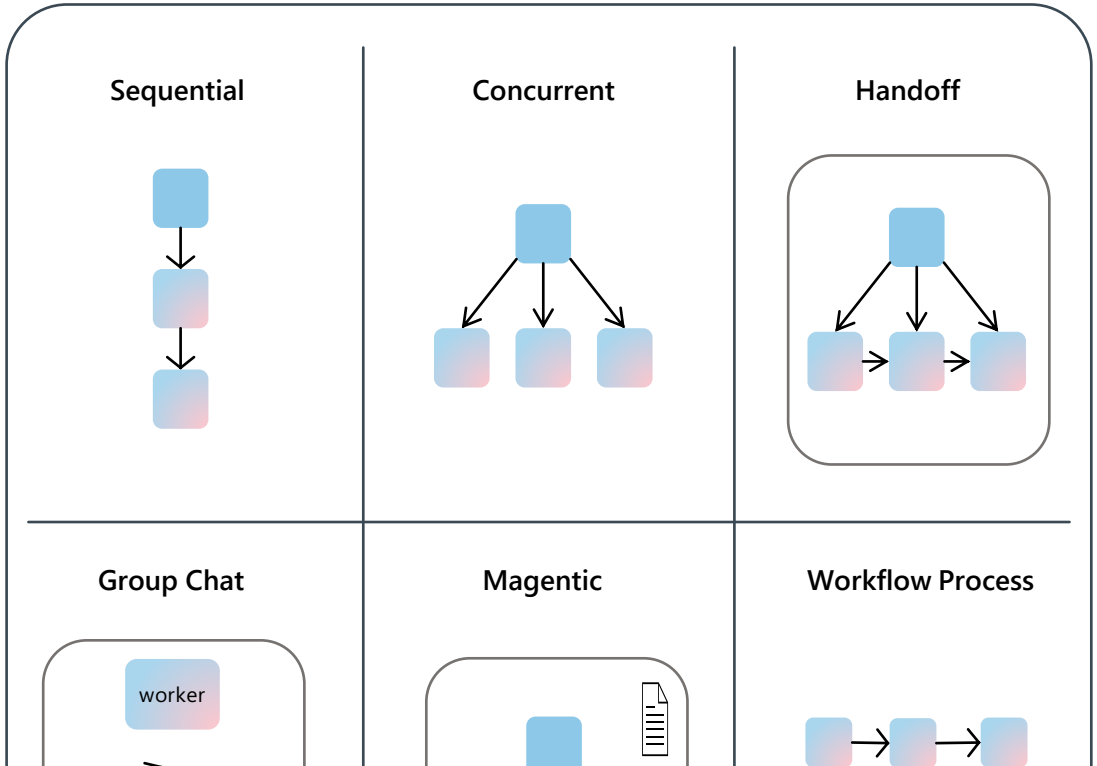
Business logic-driven orchestration

Mix of business processes and agents

Ex: Execution the Deep Research process to generate a document outline, write and refine section and synthesize into final document

Multi-Agent Orchestration

Inherit advanced orchestration patterns from Microsoft Research's AutoGen combined with Semantic Kernel's durable workflow orchestration. Prototype experimental patterns locally, then scale them confidently in production.

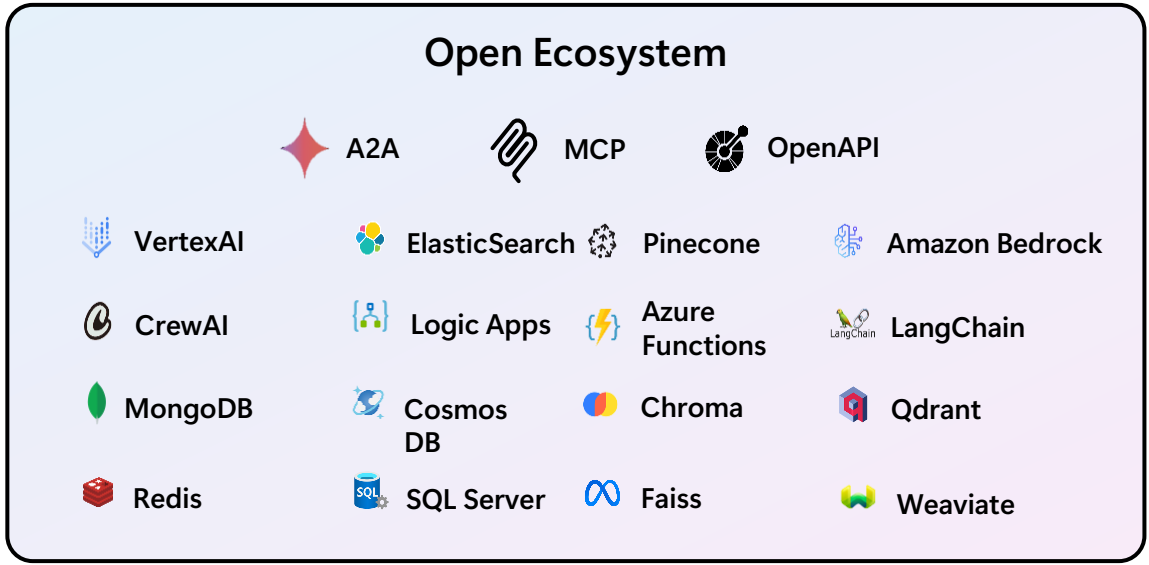


Tools and Extensibility

Functions, APIs, and MCP servers can all be turned into tools.

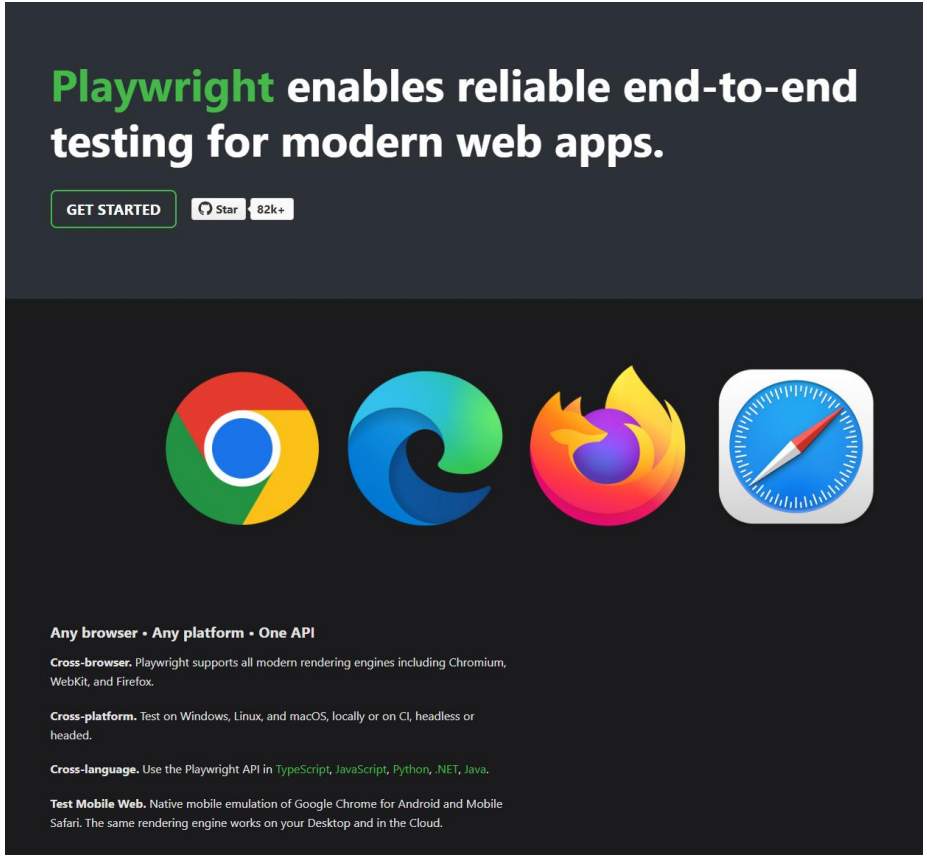
Out-of-the-box integrations to common enterprise systems, databases, and SaaS APIs — so developers don't start from scratch.

Developers can declaratively define agents in YAML and specify which tools require human approval.



Introduction to Playwright MCP

microsoft/playwright-mcp: Playwright MCP server



Hands-on workshop

- Eight modules with Python samples on building agents
- Use Azure OpenAI authentication or ask Copilot to modify authentication for OpenAI, Anthropic, Ollama, etc.
- Finally, build an agent and test the app you developed earlier
- To get started, follow the guide at

aka.ms/AgenticAiHack

Agent Framework

AI Tools and Models

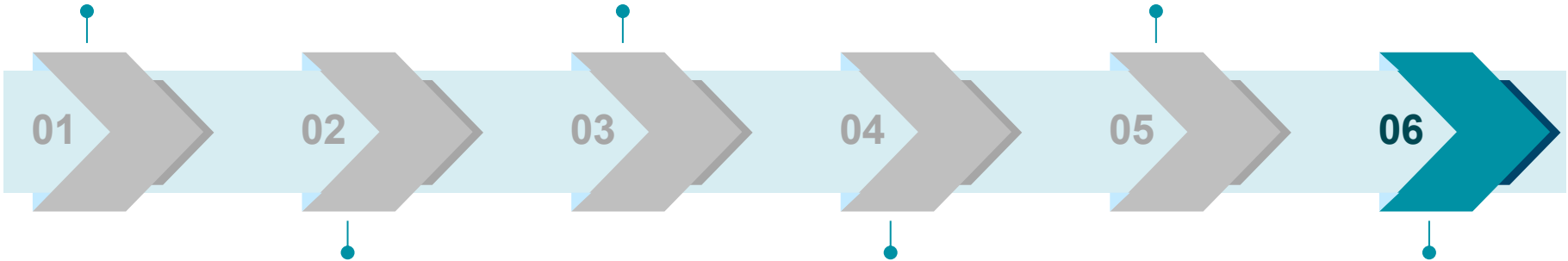
- Using AI tools and IDE integrations effectively
- Understanding model strengths and limitations

AI-Assisted Development

- Prompt engineering beyond “do X”

Spec-Driven Development

- Writing specs that are AI-readable and human-readable
- Separating intent from implementation
- Using specs as a stable contract



Vibe Coding

- Co-creating code with AI, not over-planning
- Letting AI “fill in the blanks” responsibly

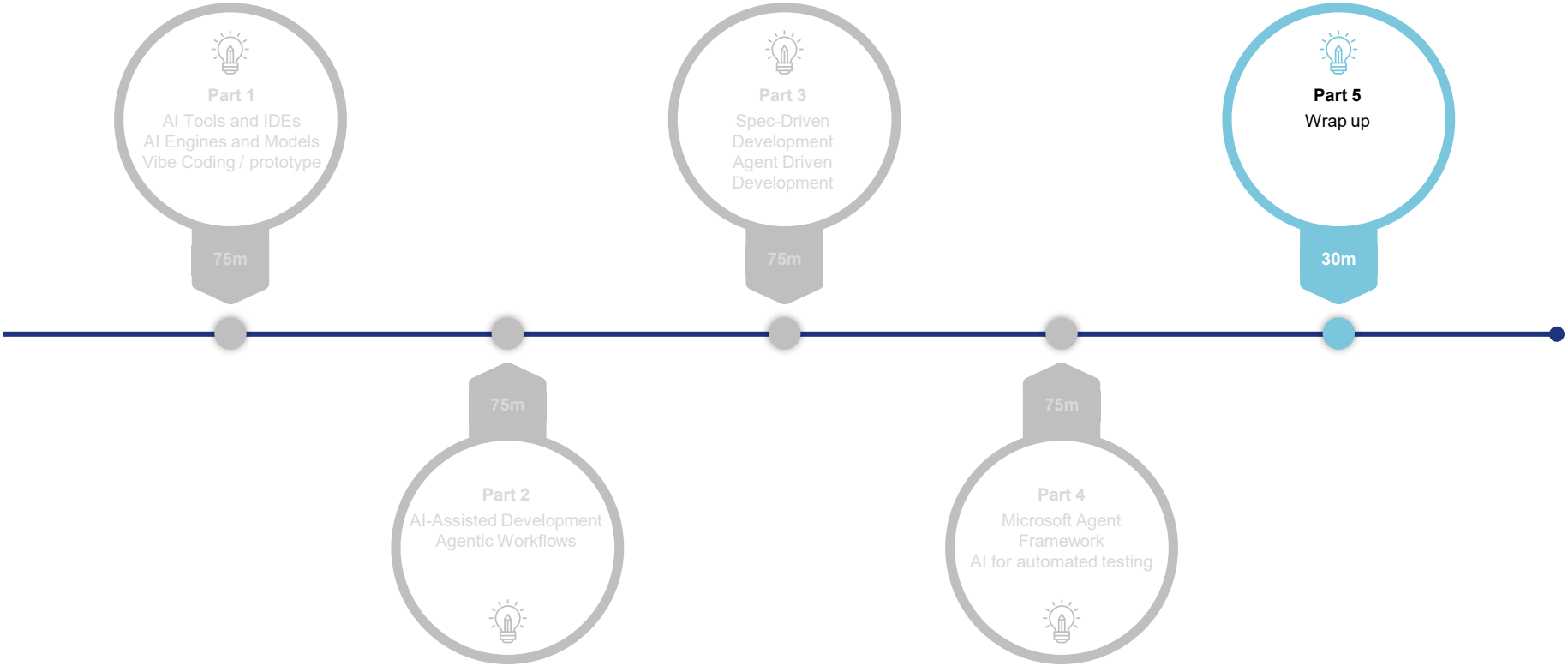
Agentic Workflows

- Agent primitives (plan, act, observe, reflect)
- Context engineering (what the AI needs to know)
- Multi-step, goal-oriented workflows

Agent Framework

- Agent framework concepts
- Tooling and orchestration
- Testing AI-driven systems
- Validating code and agent behavior

Agenda


Part 1
AI Tools and IDEs
AI Engines and Models
Vibe Coding / prototype
75m
Part 2
AI-Assisted Development
Agentic Workflows
75m
Part 3
Spec-Driven Development
Agent Driven Development
75m
Part 4
Microsoft Agent Framework
AI for automated testing
75m
Part 5
Wrap up
30m

Takeaways for this master class

Engineering shifts from coding to intent and specs

Modern AI-native development moves the engineer's focus from writing code to defining intent, specifications, architecture boundaries, and validation gates. The spec becomes the core unit of work, enabling predictable and controlled agent execution.

Reliable AI development requires structure, not ad-hoc prompts

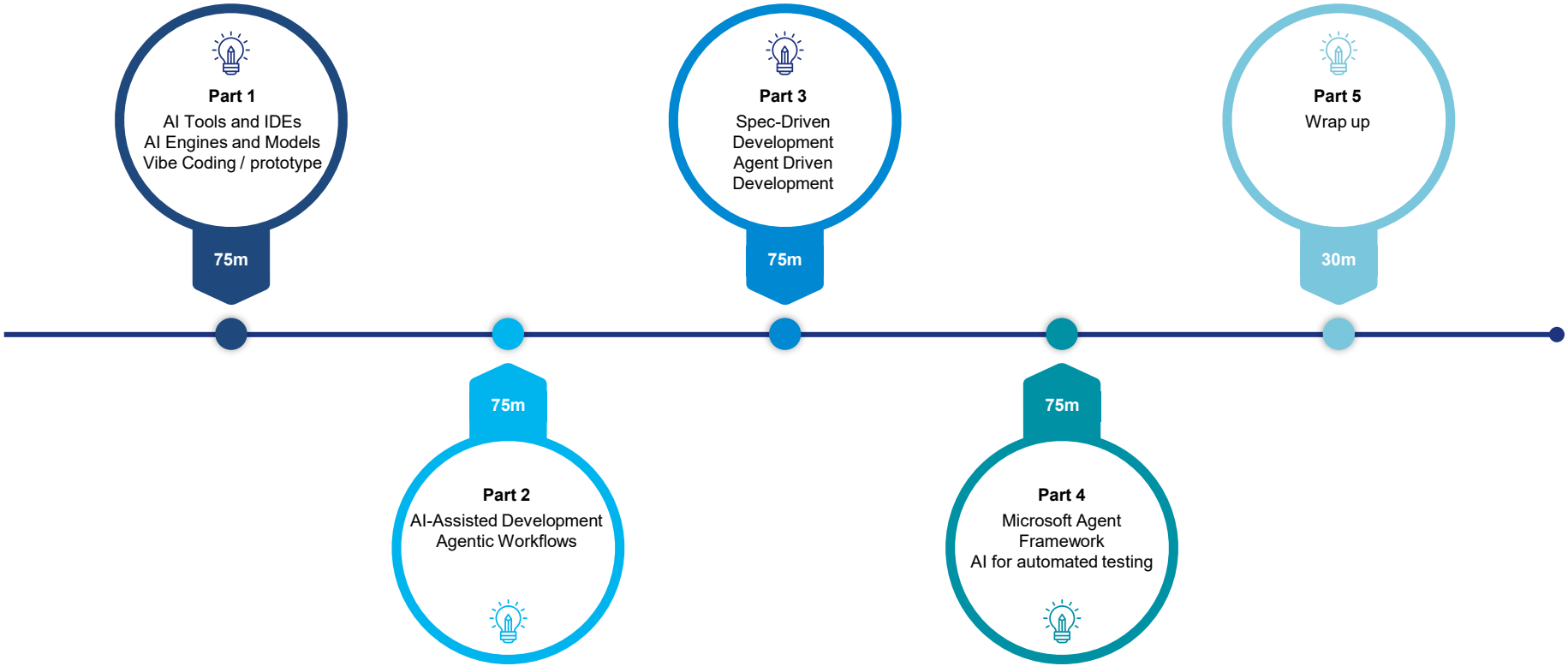
Professional AI-assisted development depends on structured prompt engineering, agent primitives, and context engineering. Treating prompts and agent behavior as reusable, versioned engineering assets makes outcomes predictable and scalable.

AI supports the full software lifecycle—not just code

AI can do more than generate code: it can drive automated testing, documentation, validation, and other lifecycle tasks. These capabilities can be orchestrated end-to-end using interoperable frameworks like the Microsoft Agent Framework, with human oversight and validation gates.

Experts Live Denmark

Agenda



Treadly Case Overview

AI Tools and Models

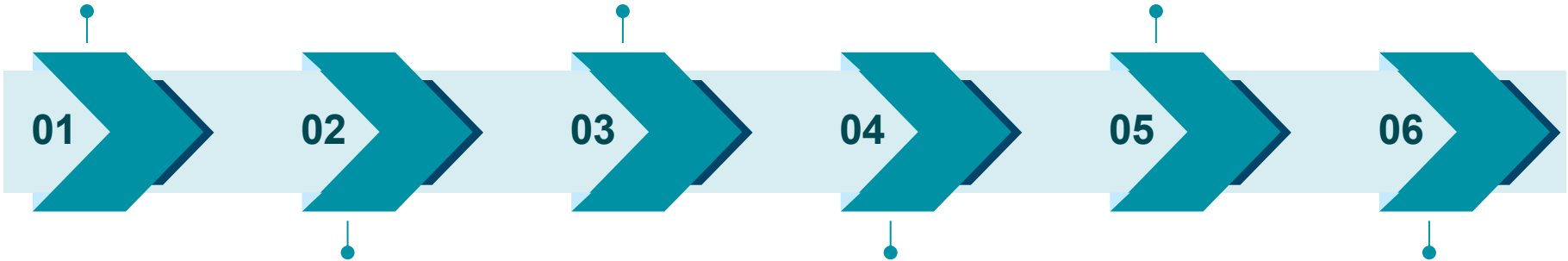
- Using AI tools and IDE integrations effectively
- Understanding model strengths and limitations

AI-Assisted Development

- Prompt engineering beyond “do X”

Spec-Driven Development

- Writing specs that are AI-readable and human-readable
- Separating intent from implementation
- Using specs as a stable contract



Vibe Coding

- Co-creating code with AI, not over-planning
- Letting AI “fill in the blanks” responsibly

Agentic Workflows

- Agent primitives (plan, act, observe, reflect)
- Context engineering (what the AI needs to know)
- Multi-step, goal-oriented workflows

Agent Framework

- Agent framework concepts
- Tooling and orchestration
- Testing AI-driven systems
- Validating code and agent behavior

DEMO

T-shirt shops

Session Feedback – THANK YOU ☺

Please evaluate this session using QR Code **during next 30 min**

- or -

Make your session feedback on your way out of the room

https://smiley.link/CFCJPFCG