Scale Your Intelligence

From MAGE to durable principles for working with AI agents

David I. Inouye

Scale Your Intelligence

AI scales the quality of your judgment

> **AI and agents can scale your intelligence—but the results will reflect the quality of your judgments.**

- Weak understanding and poor discernment can scale confusion, drift, or harm.
- Strong understanding and careful judgment can scale useful work.
- The human remains central: choose values, define what is good, shape the environment, and make consequential tradeoffs.

Goal: AI should optimize your thinking—not replace it.

MAGE Case Study

Scaling reliable agentic software engineering

Chat gathers and synthesizes information; agents change the work

Mode	Best suited to	Concrete examples
AI chat	Simple information gathering, explanation, summarization, and one-off synthesis	Claude chat; ChatGPT chat
AI agent	Inspecting artifacts, using tools, editing files, running checks, and pursuing a multi-step outcome	Claude Code; Codex

- Chat makes repeatable processes difficult because the work remains inside a conversation.
- Use chat for bounded information tasks—not sustained project work.
- Move consequential, multi-step work into an agentic environment.

Cheap code changes the engineering bottleneck

- Coding agents can produce and revise implementation extremely quickly.
- In a large system, more output creates more interactions, decisions, and possible failures.
- Requirements, architecture, evidence, deployment, maintenance, and accountability remain.

Implementation capacity can grow faster than the judgment needed to direct it.

Paraphrase of Davis, *MAGE*, pp. 4, 18–23

Discussion: what lies between speed and certainty?

Vibe coding

- Move at breakneck speed
- Judge by visible output
- Accept hidden uncertainty

Line-by-line inspection

- Manually validate every change
- Seek maximum direct oversight
- Give up much of the speed advantage

Is there a third approach that preserves both leverage and reliability? What would it require?

Discussion framing by Inouye; engineering problem informed by Davis, *MAGE*, Part 1

Scale creates a reasoning-horizon problem

- A large system exceeds what any one human or agent can actively reason over.
- Larger context windows and retrieval move the boundary; they do not remove finite reasoning.
- Without useful external representations, each new task reconstructs system knowledge from code and history.

> **Repeated reconstruction creates churn—even when each local change looks plausible.**

Paraphrase of Davis, *MAGE*, pp. 28–45

MAGE answers the scaling problem with an engineering method

MAGE = **Model-Based Agentic Software Engineering**, a method developed by James C. Davis for engineering large software systems with coding agents.

> **Model consequential knowledge → give obligations authority → convert recurring lessons into engineering capital**

Davis, “Model-Based Agentic Software Engineering,” 2026 · Davis, *Model-Based Agentic Software Engineering*, pp. 4–6

1: Modeling makes intent explicit at useful scales

> “A model is a purposeful reduction of a system. It preserves the information one engineering question needs and suppresses the detail that question does not.” — James C. Davis, *MAGE*, p. 38

- **Explicit:** Make engineering knowledge and intent explicit.
- **Compact:** Suppress details that do not matter at that level or for that question.
- **Understandable:** Make the representation usable by both people and machines.

Definition and Modeling Principle from Davis, *MAGE*, §2.1.1, p. 38 and Part 2

1: Modeling uses different models for different questions

Engineering question	Example model
How is the system divided and connected?	Architecture, component, or dependency model
What can happen, and in what order?	State machine, workflow, scenario, or use-case model
What must or must not happen?	Policy, invariant, permission, privacy, or security model
What counts as acceptable evidence?	Measurement, metric, threshold, or assurance model
Who owns a decision or artifact?	Ownership, decision, or provenance model
How do facts connect across the system?	System knowledge graph

Examples adapted from Davis, *MAGE*, Part 2

2: Alignment gives modeled obligations authority

Models can state what should hold, but representation alone does not make the work follow the model. **Alignment gives selected obligations consequence.**

Mechanism	Role	Simple example
Constraint	Restrict action	A sandbox forbids a dangerous operation
Sensor	Produce evidence	A scan reports exposed endpoints
Validator	Judge evidence	A test checks required behavior
Gate	Control admission	CI blocks a release that fails checks

Agents remain free within permitted bounds; work that violates a governed obligation does not pass.

Paraphrase of Davis's Alignment Principle: Davis, *MAGE*, pp. 4–6 and Part 3

3: Governance conversion lets judgment accumulate as engineering capital

Churn

- Rediscover the same context
- Repair the same failure
- Repeat the same review comment
- Reconcile the same inconsistency

Governance conversion

- Capture the missing fact
- Improve the shared procedure
- Add the right evidence or control
- Let future work inherit the lesson

Paraphrase of Davis's governance-conversion loop: Davis, *MAGE*, pp. 4–6 and Part 4

MAGE builds a governed engineering environment

Model consequential knowledge → give obligations authority → delegate work → inspect evidence → convert recurring lessons

- **Modeling** counters reconstruction at scale.
- **Alignment** prevents probabilistic implementation from becoming its own authority.
- **Governance conversion** lets engineering effort accumulate over time.

Summary of Davis, *MAGE*, “MAGE on One Page,” p. 6

My Broader Synthesis

Scale your intelligence beyond software engineering

Effective CEOs already scale judgment through people and culture

- Focus on consequential choices: strategy, values, and tradeoffs.
- Delegate logistics, reporting, coordination, and routine decisions.
- Shape culture and feedback so others can act well without constant oversight.

Agents make this leverage accessible—but not automatically good

- Agents make delegation far cheaper and more widely available.
- Give them explicit goals and boundaries; require evidence and escalation.
- Start with bounded work, preserve lessons, and expand incrementally.

> **AI and agents can scale your intelligence—but the results will reflect the quality of your judgments.**

Offload work that does not require your attention so you can invest it where your judgment creates the most value.

Scaling intelligence requires two connected moves

> **To scale your intelligence, externalize the thinking that matters and engineer a system that can apply it reliably, test the results, and learn from corrections.**

Make your thinking explicit and structured

1. Preserve consequential context.
2. Organize it into useful models and levels.
3. Bring the right representation to each decision.

Turn judgment into a reliable system

4. Define what good means and what evidence counts.
5. Give important rules authority and preserve lessons.
6. Formalize repeatable checks where worthwhile.

Instructor synthesis by Inouye; relationship inspired by Davis's Modeling, Alignment, and governance-conversion loop in MAGE

Make Your Thinking Explicit and Structured

Pitfall: important context remains implicit and temporary

- A long chat appears to know the project because history is still nearby.
- Goals, constraints, and corrections remain soft patterns in conversation.
- A new session, agent, collaborator, or compaction cannot reliably inherit them.

> **Principle 1: Make consequential context explicit, durable, and inspectable.**
>
> - Write down important goals, constraints, definitions, and decisions.
> - Keep them in durable artifacts that people can inspect.
> - Let agents help maintain them as the work changes.

Inouye synthesis; directly inspired by Davis's Modeling Principle and governance conversion

Examples: explicit context across domains and sessions

> **Principle 1:** Make consequential context explicit, durable, and inspectable.

Software engineering	Research	Writing
Store goals, interfaces, invariants, and decisions beside the code.	Preserve the research question, assumptions, inclusion rules, and evidence ledger.	Preserve audience, purpose, thesis, source notes, and editorial decisions.
Let the agent maintain these artifacts as the project changes.	Let later analyses inherit the same definitions and exclusions.	Let each revision inherit the same argument and citation constraints.

Examples by Inouye; principle inspired by Davis, *MAGE*, Parts 2 and 4

Pitfall: the agent must infer the shape of good work

- “Make it better” provides no stable decomposition or success surface.
- Reasoning over the whole system at once overloads both humans and agents.
- Local changes drift when their relationship to larger intent is unclear.

> **Principle 2: Build structure at useful levels of abstraction.**
>
> - Decompose the goal into connected levels or views.
> - Reason locally at the level where the decision belongs.
> - Preserve intent and interfaces between levels.

Inouye synthesis; directly inspired by Davis’s reasoning-horizon argument and Modeling Principle

Examples: abstractions for local reasoning

> **Principle 2:** Build structure at useful levels of abstraction.

Software engineering	Research	Writing
Goal → architecture → pipeline → module → code	Question → constructs → hypotheses → study → analysis	Purpose → argument → sections → paragraph claims → sentences
Decide locally while preserving interfaces between levels.	Refine one level without silently changing the research question.	Revise prose locally without losing the role of the section.

Examples and cross-domain hierarchy by Inouye; starting point: Davis, *MAGE*, Part 2

Pitfall: context is either missing or indiscriminate

- Too little context makes the agent guess.
- Dumping the whole project into every task hides what matters.
- Low-level work may optimize locally while violating a system-level goal.

> **Principle 3: Deliver the right context where each decision occurs.**
>
> - Supply the relevant goals, constraints, interfaces, and evidence.
> - Avoid flooding every task with the entire project.
> - Retrieve adjacent or higher-level context when a boundary is crossed.

Inouye extension of Davis's purposeful modeling and connected system knowledge: Davis, *MAGE*, Parts 1.3 and 2.8

Examples: dynamic context at decision time

> **Principle 3:** Deliver the right context where each decision occurs.

Software engineering	Research	Writing
A privacy edit receives the product goal, privacy invariant, module contract, and related tests.	An analysis receives the hypothesis, data dictionary, preprocessing decisions, and relevant prior results.	A section revision receives the audience, thesis, outline, neighboring claims, and source constraints.

The worker can request adjacent or higher-level context when a local decision crosses a boundary.

Dynamic-context formulation and examples by Inouye; informed by Davis, *MAGE*, Parts 1.3 and 2.8

Turn Judgment into a Reliable System

Pitfall: capability is blamed for underspecified work

- “Build a dashboard,” “find an interesting result,” and “improve this draft” are not decisions.
- A strong model can produce many plausible outputs without knowing which one matters.
- Faster creation makes it easier to travel quickly in the wrong direction.

> **Principle 4: Engineer the environment around value, requirements, and evidence.**
>
> - Define stakeholder value before choosing features or methods.
> - Turn “good” into requirements and acceptance evidence.
> - Keep human authority over consequential ambiguity and tradeoffs.

Inouye synthesis; related to Davis’s changed-bottleneck argument and governed engineering environment

Examples: defining what good means

Principle 4: Engineer the environment around value, requirements, and evidence.

Software engineering	Research	Writing
Name the user, task, constraints, failure costs, and acceptance evidence before choosing features.	Name the contribution, competing explanations, falsifying evidence, and acceptable uncertainty.	Name the reader, desired effect, central claim, evidence standard, and tone before polishing prose.

Cross-domain formulation and examples by Inouye; MAGE influence: Davis, *MAGE*, pp. 4–6

Pitfall: good intentions lack authority

- Policies are remembered inconsistently.
- The same failure is corrected repeatedly.
- Review occurs late, after the cost of change has increased.

> **Principle 5: Govern invariants and turn recurring judgment into durable structure.**
>
> - Give important invariants mechanisms that can affect the work.
> - Preserve consequential decisions and their rationale.
> - Convert repeated corrections into instructions, checks, or gates.

Inouye generalization of Davis's Alignment Principle and governance conversion

Examples: governing durable judgment

Principle 5: Govern invariants and turn recurring judgment into durable structure.

Software engineering	Research	Writing
Record architecture decisions; gate releases on privacy and test evidence.	Preserve protocol decisions, provenance, exclusions, and deviations; require evidence before claims advance.	Archive sources and exact quotations; enforce citation and editorial checks before publication.

Humans retain novel tradeoffs. Repeated judgments become instructions, models, validators, or gates that future work inherits.

Cross-domain examples by Inouye; directly inspired by Davis, *MAGE*, Parts 3–4

Pitfall: probabilistic judgment is used where a check would suffice

- AI repeatedly re-counts, re-parses, or re-checks exact properties.
- Soft review varies between runs and may miss simple violations.
- Natural-language intent and implementation silently diverge.

> **Principle 6: Make the process deterministic and formal wherever the value exceeds the cost.**
>
> - Script exact, repeatable checks instead of asking AI each time.
> - Formalize important properties when the benefit justifies the upkeep.
> - Reserve AI and human review for ambiguity, coherence, and tradeoffs.

Determinism emphasis by Inouye; informed by Davis's constraints, sensors, validators, and gates

Examples: dividing formal and AI judgment

> **Principle 6:** Make the process deterministic and formal wherever the value exceeds the cost.

Software engineering	Research	Writing
Use schemas, types, tests, dependency checks, and reproducible builds; reserve AI review for architecture and novel bugs.	Script transformations and statistical checks; use structured metadata; reserve AI synthesis for interpretation and competing explanations.	Check links, quotations, citations, lengths, and required sections mechanically; reserve AI review for coherence and reader impact.

Maintain human-readable and machine-readable views when each supports a different kind of reasoning—and check that they correspond.

Cross-domain division of labor by Inouye; MAGE influence: Davis, *MAGE*, Part 3

Effective AI work is an engineered learning loop

Make intent explicit → structure the work → deliver relevant context → delegate → observe evidence → govern → preserve the lesson

The goal is not to eliminate ambiguity. It is to expose consequential ambiguity early enough for human judgment—and to stop paying for the same judgment twice.

Inouye synthesis; adapted from Davis's Modeling, Alignment, and governance-conversion loop

Apply the loop to one real project

Choose a current software, research, writing, or AI Product task:

1. What important fact or judgment exists only in your head or chat history?
2. What representation would let both you and an agent reason about it?
3. What evidence would distinguish success from plausible-looking failure?
4. Which repeated check or correction should become durable?

Exercise by Inouye; informed by Davis's MAGE workflow

Continue with the primary sources

- James C. Davis, “Model-Based Agentic Software Engineering (MAGE)”, 2026.
- James C. Davis, *Model-Based Agentic Software Engineering*, evolving online edition, 2026.