Scale Your Intelligence: Six Operating Principles

ECE 57000 — August 31, 2026

David I. Inouye

Friday introduced MAGE as a method for scaling reliable work

- Cheap implementation moves the bottleneck toward direction, judgment, and assurance.
- **Modeling:** make consequential intent explicit at useful scales.
- **Alignment:** give selected obligations authority through constraints, evidence, validation, and gates.
- **Governance conversion:** turn recurring lessons into durable engineering capital.

Model consequential knowledge → give obligations authority → delegate work → inspect evidence → preserve the lesson

Discussion: a fast agent exposes a governance problem

An AI Product team asks a coding agent to add appointment reminders. The tests pass and the demo looks good. During the pilot, a reminder exposes the clinic type on a shared lock screen. The privacy need had appeared in an early stakeholder interview but was never captured in the product requirements.

> **Before the next feature, what should the team change so this privacy need guides both the agent's work and the release decision?**

- What should be written down and connected to the design?
- What evidence or check should be required before release?
- How should this incident improve future work?

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

Discussion: product direction needs durable structure

A team is choosing between a scheduling assistant and a risk dashboard. It has interview evidence and a one-page template from an earlier product with fields for the goal, target users, candidate features, and owner. However, the team has not recorded its current priorities, decision criteria, or reasons for choosing between the features.

> **What 3–4 things should the team include in its one-page product model so it can make and explain this decision?**

What can it reuse from the earlier template—and what must be adapted?

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

Discussion: a strong pilot hides a control decision

A campus-facilities system reads maintenance reports, assigns urgency, and drafts work orders. Its pilot metrics look strong, but it classified one gas-odor report as routine because text inside an attachment was never extracted.

> **Before expansion, design the minimum release gate that would catch this failure.**

- What evidence must be present before automatic dispatch?
- When must a human decide?

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