Paper PM04

NeuroDriven Mentorsip: Recognizing and Teaching Different Kinds of Thinkers

Ellada Abrahamyan, STATECS LLC, Yerevan, Armenia
Arman Melkonyan, STATECS LLC, Yerevan, Armenia

ABSTRACT

This paper presents a cognitive-informed approach to mentoring clinical programmers, grounded in both psychological theory and field experience. It defines six thinking styles—linear, visual, pattern-driven, conceptual, experimental, and emotionally attuned—that shape how individuals learn, code, and respond to feedback. The method equips mentors to identify each style through behavioural cues and problem-solving habits, and provides targeted guidance for teaching, communication, and support. It also addresses common learning barriers unique to each type. Designed to challenge the default, one-size-fits-all mentorship model, this approach helps mentors connect with how their mentees think—not just what they know. By speaking the brain's language, mentors can foster learning that is not only faster and more effective, but also more enduring—empowering mentees to grow with confidence and building teams that are cognitively supported, emotionally engaged, and prepared to grow with purpose.

INTRODUCTION

Mentorship plays a central role in the development of clinical programmers, where the pace of work and the demands of regulatory compliance make learning both complex and essential. Traditional mentorship models, however, often assume that all mentees process information and respond to guidance in the same way. This one-size-fits-all approach risks overlooking the cognitive differences that shape how programmers interpret tasks, approach problem-solving, and integrate feedback. Misalignment between a mentor's teaching style and a mentee's thinking style can slow progress, increase frustration, and limit the long-term effectiveness of training.

This paper proposes a cognitive-informed mentorship framework that recognises and leverages cognitive diversity within clinical programming teams. Drawing from psychological theory and field experience, six distinct thinking styles are described: linear, visual, pattern-driven, conceptual, experimental, and emotionally attuned. These are not rigid categories, but practical archetypes that highlight differences in how individuals learn and apply knowledge. By equipping mentors to identify these styles and adapt their communication, teaching, and feedback strategies accordingly, the framework aims to improve the efficiency, confidence, and resilience of both individual programmers and the teams they support.

Literature Review

Psychological theory provides a complementary perspective by emphasising cognitive diversity. Kolb's (1984) model of experiential learning and Gardner's (1983) theory of multiple intelligences both illustrate that individuals process and apply knowledge in fundamentally different ways. These frameworks suggest that learning styles shape not only what is learned but also how learners respond to instruction and feedback. Yet in clinical programming literature, mentorship guidance remains largely content-driven, with little integration of cognitive frameworks. This gap signals an opportunity: applying established psychological insights to mentorship practices in clinical programming may enable approaches that are more adaptive, effective, and aligned with how programmers actually think and work.

Methodology

The framework was developed by combining psychological theory with direct mentoring experience in clinical programming teams. Established work in learning and cognition provided the foundation for recognising that individuals process and apply knowledge in different ways. These theoretical insights were then compared with practical observations of how programmers approached specifications, structured code, and responded to feedback.

Through an iterative process, recurring patterns were grouped into six functional archetypes: linear, visual, pattern-driven, conceptual, experimental, and emotionally attuned. These archetypes are not rigid categories, but working constructs that capture common tendencies in how mentees organise information, engage with tasks, and adapt to guidance. The final set was chosen because it consistently reflected observable differences in learning pace, responsiveness to instruction, and preferred modes of feedback, while remaining simple enough for mentors to apply in practice.

NeuroDriven Quiz Implementation

To operationalize this framework, a digital self-assessment tool was developed and administered to clinical programming teams.
The quiz consisted of scenario-based questions designed to capture cognitive tendencies through realistic work situations, such as interpreting incomplete specifications, prioritizing under time pressure, or designing study-level macros. Responses were analyzed to assign each participant a dominant cognitive style aligned with one of the six archetypes.

While management and mentorship leads were often able to intuitively recognize thinking patterns through day-to-day observation, the quiz provided a structured, data-supported validation of these assessments. Comparative analysis confirmed a high level of consistency between quiz outcomes and managerial evaluations, reinforcing the model's practical accuracy and applicability in real-world team environments.

Main Body

Linear Thinkers

Identification Cues

- Process-oriented questions: They ask about the order of tasks rather than the logic behind them, e.g., "What should I do first?" or "Which step comes next?"
- Difficulty with incomplete information: When instructions are partial, they pause or seek clarification rather than filling gaps themselves, often circling back to earlier steps before progressing.
- Sequential working style: They prefer completing one task fully before starting the next and may resist parallel workstreams.
- Strong attention to order and detail: They are meticulous about ticking off requirements line by line and confirming nothing has been skipped.

Teaching Strategies

- Provide ordered workflows: Break down tasks into defined sequences, showing how each stage depends on the one before it.
- Segment complex tasks: Divide large deliverables into smaller, manageable stages so progress can be validated incrementally.
- Reinforce with structured tools: Written checklists, flow diagrams, and templates give them a concrete sense of direction.
- Maintain checkpoints: Pause between stages to confirm understanding before moving forward.
- Anchor in standards: SOPs, programming conventions, and specifications provide reassurance, as they align with the rules these mentees trust.

Feedback Approach

- Be concrete: Specify the step where the issue occurred and explain why it matters.
- Maintain sequence: Deliver feedback in the order they followed the task, so corrections map directly onto their process.
- Highlight cause and effect: Show how an error in one stage leads to downstream issues, reinforcing linear logic.
- Encourage checkpoints: Suggest self-review at natural breakpoints to prevent late-stage surprises.
- Balance correction with reassurance: Because deviations unsettle them, affirm what they executed correctly before pointing out what to adjust.

Strengths

- Excel at structured programming tasks that follow clear specifications and standards.
- Strong attention to detail and order reduces risk of overlooked requirements.
- Well-suited for documentation-driven deliverables (e.g., traceability, standards reviews) where sequence and completeness are critical.

Pitfalls

- Context switching: If introduced to a project mid-stream (e.g., starting directly with TLFs), they may need extra time to re-establish order by revisiting protocols and specifications.
- Ad-hoc requests: Disruptive because these lack predefined sequence and force them to create structure on the fly.
- Rigidity: May insist on following tasks strictly “in order,” even when efficiency requires skipping ahead or working with draft inputs.

Visual Thinkers

Identification Cues

- Ask for diagrams or layouts: Instead of asking “What comes first?”, they ask “Can you show me how this connects?”
- Think spatially: They imagine datasets, processes, and outputs as interconnected blocks rather than linear steps.
- Use visual language: Their questions include phrases like “I need to see the bigger picture” or “How does this dataset fit into the flow?”
- Seek relationships: They are quick to spot links across deliverables and often sketch them on paper or whiteboards.
- Struggle with text-heavy instructions: Lengthy, word-based specifications may overwhelm them unless translated into diagrams or highlighted structures.

Teaching Strategies

- Create images they can picture: Frame processes in terms of vivid scenarios that they can imagine and retain.
- Demonstrate with real artifacts: Show them an actual Case Report Form (CRF) instead of explaining what a CRF is, or display a sample protocol and point out a section rather than describing the structure verbally.
- Draw diagrams while teaching: Use a blackboard, whiteboard, or shared screen to sketch dataset flows, derivation paths, or TLF shells. Making abstract processes visible helps them grasp connections quickly.
- Encourage their own sketching: Ask them to diagram how they see data moving or how deliverables connect. Their drawings reveal both strengths in understanding and areas needing clarification.
- Pair words with visuals: When discussing specifications or outputs, highlight sections directly on documents, use color-coded notes, or mark-up shells so they see structure as you explain it.

Feedback Approach

- Restore orientation: Instead of pointing to an error in isolation, explain how it shifts the balance of the overall structure.
- Emphasize placement: Frame corrections in terms of where something belongs, e.g., “this element needs to sit here so the flow holds together.”
- Make contrasts visible: Place the expected and actual side by side so they can immediately see the difference.
- Encourage visual check-ins: Ask them to redraw or outline their process, then guide them to notice where it diverges.
- Reassure through coherence: Stress that one adjustment can bring the whole structure back into alignment, which restores their confidence.

Strengths

- Quickly grasp the big picture, seeing how datasets, outputs, and processes fit together
- Strong at traceability reviews, where visualizing flows ensures every result connects back to its source.
- Skilled at clarifying complexity for others by sketching or mapping out workflows.
- Bring creativity in framing problems, offering alternative perspectives when standard explanations do not resonate.

Pitfalls

- Overlooking fine detail: In focusing on the big picture, they may miss subtle compliance issues or technical requirements.
- Struggle with text-heavy specifications: Dense, word-based documents without diagrams slow their progress.
- Risk of over-simplification: They may assume a connection exists where it does not, creating shortcuts that undermine accuracy.
- Disoriented by missing visuals: Without diagrams, CRFs, or structural cues, they can lose confidence or hesitate to move forward.

Pattern-Driven Thinkers

Identification Cues

- Notice repetition instinctively: They ask questions like “Haven’t we done this before?” or “Is this variable handled the same way in another domain?”
- Seek general rules: Instead of focusing on single cases, they look for patterns they can apply broadly.
- Think in terms of reusability: They prefer to solve problems once and create code, macros, or workflows that can be reused.
- Draw parallels across tasks: They connect current assignments with earlier projects or other domains.
- Lose patience with exceptions: When rules don’t hold, they may feel frustrated or stuck.

Teaching Strategies

- Frame learning in terms of rules: Explain not just what to do but why it fits a larger pattern.
- Emphasize reusability: Show how one solution applies to multiple contexts, reinforcing their instinct to generalize.
- Encourage abstraction: Challenge them to identify similarities across domains, outputs, or code.
- Provide opportunities to standardize: Assign tasks like macro development, standards checks, or consistency reviews.
- Balance rules with context: Remind them that exceptions exist and must be handled deliberately.

Feedback Approach

- Validate their logic: Acknowledge the correctness of the pattern they applied before pointing out the exception.
- Highlight the boundary of the rule: Show them the exact point where their generalization no longer holds. Frame it as a natural “edge case,” not a failure.
- Use contrast: Place a case that fits their rule next to one that doesn’t. Seeing the difference helps them adjust without feeling their whole approach was wrong.
- Encourage flexibility: Phrase corrections as adaptations, e.g., “The rule works most of the time, but here’s how we adjust when it doesn’t.”
- Anchor in standards vs. exceptions: Reinforce which patterns are stable (e.g., CDISC conventions, company macros) and which require context-specific judgment.

Strengths

- Highly efficient when applying reusable code, macros, or templates.
- Skilled at maintaining consistency across large deliverables.
- Strong at recognizing redundancy and simplifying workflows.
- Add value in standardization efforts, where repeatability is essential.

Pitfalls

- Over-generalization: Risk of assuming all cases follow the same pattern, leading to errors when exceptions are ignored.
- Frustration with irregularity: May stall when confronted with outliers or tasks that do not follow expected rules.
- Risk of rigidity: Can prioritize rules over context, missing study-specific nuances.
- Over-focus on efficiency: In seeking reusable solutions, may overlook details that require customization.

Conceptual Thinkers

Identification Cues

- Ask “why” before “how”: They want to understand the rationale behind a task before beginning execution.
- Seek underlying principles: Instead of looking at one dataset, they ask how it connects to study objectives or trial design.
- Use abstract language: They talk in terms of ideas and categories rather than line-by-line instructions.
- Value theory: They are quick to reference methodologies, standards, or models as the foundation for their work.
- Lose interest in isolated detail: They may disengage if given repetitive tasks without context.

Teaching Strategies

- Begin with context: Explain the purpose of a deliverable in relation to trial objectives or regulatory expectations.
- Use principles, not just steps: Show how a programming rule or derivation reflects a methodological standard, rather than teaching it as a routine.
- Encourage discussion: Let them ask “why” questions and engage in dialogue about meaning.
- Map ideas to practice: After explaining the theory, guide them in applying it to a concrete task.
- Challenge with conceptual tasks: Assign work such as defining standards, improving specifications, or explaining rationale in documentation.

Feedback Approach

- Speak to their need for coherence: Use language that appeals to structure and intent. For example, instead of “this is incorrect”, say “this choice doesn’t fully reflect the study objective”.
- Reframe errors as gaps in reasoning: Present mistakes as incomplete logic, e.g., “your approach covers part of the principle, but it leaves out this condition.”
- Invite them into dialogue: Ask reflective prompts such as “how does this align with the endpoint definition” or “what principle guided your choice here.”
- Keep feedback future-oriented: Emphasize growth, e.g., “next time, think about how this rule extends to edge cases.”

Strengths

- Provide strong contextual understanding, ensuring deliverables align with study purpose and design.
- Good at explaining the rationale behind programming choices to stakeholders.
- Contribute to standards development and methodological consistency.
- Bring innovation by questioning assumptions and suggesting conceptual improvements.

Pitfalls

- Overlooking detail: May miss technical errors or compliance issues when focused on theory.
- Slow execution: Spend extra time understanding context before coding, delaying output.
- Risk of over-analysis: Can get stuck debating principles instead of delivering practical solutions.
- Frustration with rote work: May disengage when tasks feel repetitive or disconnected from purpose.

Experimental Thinkers

Identification Cues

- Jump into coding quickly, preferring to test ideas rather than plan extensively.
- Ask exploratory questions such as “what happens if we try this” instead of “what is the correct step.”
- Show curiosity for alternatives, often running multiple approaches to compare outcomes.
- Appear restless with too much upfront instruction, preferring to learn through direct engagement.

- Take setbacks as part of the process, sometimes overlooking efficiency in favor of exploration.

Teaching Strategies

- Allow room for testing: Give them tasks where they can try multiple approaches without fear of failure.
- Provide sandboxes: Offer safe, smaller datasets or practice environments for trial-and-error learning.
- Encourage reflection after testing: Ask them what worked and why, so learning consolidates into principles.
- Guide them toward efficiency: Help them balance exploration with structure by showing when experimentation adds value and when it wastes time.
- Frame lessons as challenges: They respond well to open-ended prompts like “find three different ways to solve this” or “optimize this process.”

Feedback Approach

- Normalize missteps: Use phrases like “this result gives us useful information” instead of “this was wrong.”
- Highlight learning value: Say “your test showed why this approach doesn’t fit, which is important to see.”
- Redirect exploration: Frame corrections as guidance, e.g., “your idea works in some cases, let’s narrow it so it applies consistently.”
- Reinforce curiosity: Affirm their strength with phrases like “I like how you tested multiple paths” before guiding them back to efficiency.
- Keep tone future-focused: Encourage iteration by saying “next time, adjust your test to include…” so feedback feels like fueling the next experiment.

Strengths

- Highly adaptive in unfamiliar or evolving situations.
- Quick to generate alternatives when standard methods fall short.
- Resilient to mistakes, seeing them as opportunities for discovery.
- Contribute creativity and innovation to problem-solving within teams.

Pitfalls

- Risk of inefficiency from excessive trial-and-error without enough structure.
- May overlook standards or conventions in favor of novel approaches.
- Can become scattered if exploration lacks boundaries.
- Frustration may arise when constrained by rigid processes.

Emotionally Attuned Thinkers

Identification Cues

- Ask questions that check for approval, such as “is this okay” or “am I on the right track.”
- Notice tone and body language quickly, responding strongly to encouragement or criticism.
- Work best in supportive environments where they feel valued.
- Show hesitation or self-doubt if feedback feels harsh or dismissive.
- Express motivation when their work is recognized and connected to team impact.

Teaching Strategies

- Establish psychological safety: Make it clear that mistakes are part of learning and will not be judged harshly.
- Use affirmations strategically: Reinforce progress with simple validation, e.g., “this step is solid.”
- Personalize teaching: Relating examples to their role in the team or project keeps them engaged.
- Break down criticism gently: Wrap corrections in supportive framing so they do not interpret mistakes as personal failure.
- Encourage collaboration: They often thrive when working in pairs or groups where emotional support is built in.

Feedback Approach

- Begin with affirmation: Say “you’re on the right path here” before introducing corrections.
- Keep feedback relational: Use phrases like “let’s adjust this together” instead of “you need to fix this.”
- Emphasize growth over error: Frame issues as part of development, e.g., “this is a step we can strengthen” rather than “this is wrong.”
- Match tone to encouragement: Maintain warmth and calmness so corrections feel constructive, not critical.
- Reinforce team value: Remind them how their contribution supports the larger group, which keeps motivation high.

Strengths

- Build strong relationships, often acting as bridges within teams.
- Highly motivated when they feel recognized and supported.
- Intuitive in understanding team morale and interpersonal challenges.
- Bring empathy and cohesion to technical environments that can otherwise feel impersonal.

Pitfalls

- May take criticism personally, leading to reduced confidence
- Can become dependent on external validation instead of building self-assurance.
- Risk of avoiding conflict, even when it would clarify misunderstandings.
- Productivity may suffer if emotional safety is compromised.

FINDINGS

Implementation of the NeuroDriven Mentorship framework through the NeuroDriven Quiz yielded measurable improvements in both technical performance and behavioural efficiency. Mentors adapted their teaching and communication methods to align with each mentee’s cognitive profile, resulting in more targeted and effective interactions.As summarised in Table 1, quantitative outcomes demonstrated clear gains across key performance indicators. Average interview and assessment scores increased from 72 to 90 (+25%), reflecting deeper comprehension and stronger problem-solving precision. The frequency of repeated explanations per mentee per month decreased from 4.0 to 1.5 (−62%), indicating greater conceptual clarity and self-sufficiency among mentees. Average learning-curve duration also improved, with mentors reporting a reduction from 25 to 18 days (−28%), suggesting faster adaptation to assigned programming areas.

Table 1. Summary of Quantitative Outcomes Following NeuroDriven Mentorship Implementation

Metric	Pre-Implementation	Post-Implementation	Change
Interview & Assessment Scores (average score / 100)	72.0	90.0	+25%
Repeated Explanations (per mentee per month)*	4.0	1.5	−62%
Average Learning Curve Duration (days)**	25.0	18.0	−28%

* Values derived from mentor self-reports collected via a brief internal survey assessing how often key concepts had to be re-explained to the same mentee within a one-month period.
** Represents mentor-reported estimates of the average time required for mentees to become independently productive within their assigned programming area. The precise task topics varied across teams but were consistent within each mentor’s response set.

CONCLUSION

The NeuroDriven Mentorship framework demonstrates that cognitive awareness can be applied as a measurable factor in professional development within clinical programming. By integrating the six identified thinking styles into mentorship design, teams achieved quantifiable improvements in learning speed, communication efficiency, and retention.

The introduction of the NeuroDriven Quiz provided a structured method to validate cognitive profiles that mentors often recognized intuitively, confirming that behavioral observation and data-driven assessment can complement each other effectively. This alignment strengthened confidence in the framework's practical utility.

Overall, the results indicate that mentorship informed by cognitive style fosters greater precision, adaptability, and collaboration. Beyond its immediate benefit to mentorship programs, this approach offers potential for wider applications in leadership training, onboarding, and performance management promoting organizations that think, teach, and grow with cognitive diversity in mind.

REFERENCES

Kolb, D. A. (1984). *Experiential Learning: Experience as the Source of Learning and Development.* Englewood Cliffs, NJ: Prentice Hall.

Gardner, H. (1983). *Frames of Mind: The Theory of Multiple Intelligences.* New York: Basic Books.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Ellada Abrahamyan
Company: STATECS LLC
Email: ellada.abrahamyan@statecs.com
Website: https://www.statecs.com/