Bloomberg Tax
Tax Management Memorandum™

January 23, 2026

AI Reshapes Software R&D Tax Credits, Eligibility Landscape

Ajay Wanchoo and Hogan Humphries*
KPMG US

AI's integration into software development challenges traditional R&D tax credit criteria, necessitating new frameworks for eligibility.

As artificial intelligence and generative AI tools become increasingly integrated into software development, questions are emerging about how these technologies impact eligibility for the research and development tax credit under §41 of the Internal Revenue Code. Developers are now using innovative approaches such as "vibe coding" and "agentic coding" that allow them to generate and refine code using natural language, fundamentally reshaping the software development process and required skillsets.

While the regulatory landscape currently lacks clear guidance or precedent on these practices, this article explores how the adoption of AI aligns with the four core criteria for R&D tax credit qualification.

By examining each test—Permitted Purpose, Technological in Nature, Elimination of Uncertainty, and Process of Experimentation—the article provides insight into how evolving methodologies may affect R&D eligibility and highlights the need for updated frameworks to address the rapidly changing environment.

* Ajay Wanchoo is a senior managing director in the Accounting Methods and Credit Services practice in NYC. Hogan Humphries is a managing director in the Washington National Tax Methods group.

Four Tests

Software developers are developing new skillsets, and the new techniques are changing the nature of the software development lifecycle. This is a quickly changing landscape and there currently is no guidance or case law addressing the topic. How the use of AI fits into the four criteria that must be satisfied for activities to qualify for the R&D tax credit is an evolving question under the traditional framework that applies – consider the four tests.

Permitted Purpose Test. This test requires that the research be undertaken to create a new or improved business component in terms of its function, performance, reliability, or quality. The introduction of AI doesn't change the *goal*, which is still to improve a product, process, technique, formula, invention, or computer software. For instance, if a team uses an AI tool to develop a more efficient database, does it change purpose—that of improving the performance and reliability of that business component? The use of AI is a change in methodology, not a change in intent.

"Technological in Nature" Test. This test requires the research to be fundamentally based on principles of engineering, computer science, or the physical or biological sciences. One could ask if a developer prompting an AI in plain English is still performing a technological activity. Could it be argued that the developer's role is evolving from a hands-on coder to a systems thinker and a director of the AI, focusing on higher-level technological pursuits such as architectural design, complex algorithm development, and performance optimization? In this view, the human is the dominant actor, using the AI as merely an advanced tool to apply principles of computer science.

Elimination of Uncertainty Test. This test requires that the research be intended to discover information that would eliminate uncertainty concerning the capability of or method for developing or improving a business component, or the appropriate design of that new or improved business component. Does AI, with its vast knowledge base, eliminate this uncertainty? An alternative view is that AI reframes it. The uncertainty may shift from the micro-level task of *writing a piece of code* to the macro-level challenge of *engineering a reliable system*. For example, instead of struggling with the correct syntax for a database query, the developer now faces the uncertainty of which of three viable, AI-generated microservice architectures will scale most effectively under unpredictable user loads.

Process of Experimentation Test. This test requires a process of evaluating one or more alternatives to achieve a result. With AI, this process arguably becomes more significant, not less. The developer, acting as a lead investigator, can now use AI to rapidly generate multiple viable alternatives that must be systematically tested, validated, refined, and potentially discarded. The experimentation moves from the slow manual process of coding each alternative to the more complex and cognitive process of designing the tests, evaluating the

AI's output, and making critical design trade-offs. The very act of testing multiple AI-generated approaches could in itself be viewed as a core process of experimentation.

A spectrum of usage patterns is emerging in how AI is leveraged for supporting the creation of software solutions. These can vary between merely accepting AI-generated code and a more rigorous, hands-on model, where the engineer retains full ownership and responsibility and hence the human is the dominant actor actioning the research process—in the real world, some hybrid or variations of these are being adopted to varying degrees. This new paradigm demands skills in problem decomposition, critical review, advanced simulation, and systems-level thinking. The disciplined, human-led, Gen AI supported approach, with its emphasis on rigorous review and advanced testing, may strengthen the case for R&D eligibility.

Viewing a Bug as a Failed Scientific Experiment

In the AI-assisted paradigm, one may see an extraordinary rise in the number of bugs being reported during the development process. In the new Gen AI-assisted software development lifecycle, would it make more sense to consider that one is not just "fixing a bug," but rather documenting the failure of a hypothesis? In this framework, the bug is the proof that the initial hypothesis—that the AI-generated code will work—was false. The debugging and testing process then becomes the **new, qualified experiment**. A relevant analogy may be found in material-world engineering.

Analogy: The Automotive Crash Test.

- **The initial hypothesis:** A team of automotive engineers uses a sophisticated computer simulation (the AI) to design a new chassis. Their hypothesis is: "This computer-generated design will be safe in a 40-mph frontal collision."
- **The experiment:** They build a physical prototype and subject it to a real-world crash test (the software integration and load test).
- **The discovery (the bug):** Upon impact, the A-pillar buckles in a way the simulation didn't predict. This isn't a simple "bug"; it is the **formal disproof of the initial hypothesis**. It is the discovery that the computer's generalized model was technologically insufficient for the complex, real-world physics of the crash.
- **The new, qualified experiment (the debugging process):** The team doesn't simply "fix the dent." They begin a new process of experimentation: analyzing high-speed camera footage, taking metallurgical samples, and forming new hypotheses, such as "Is the material too brittle?". They then build and test new iterations until a design is validated. This iterative cycle of analysis, redesign, and retesting would then be the core research activity.

Just as the automotive engineers' work shifted from the initial design to the complex process of physical testing and reinforcement, the software developer's work shifts from the initial AI prompt to the complex process of integration testing, debugging, and architectural refactoring required to make the AI's output robust in a real-world system. By framing documentation around this concept, one may argue that the iterative "prompt-select-refine-test-debug" loop is transformed from a series of routine tasks into a clear and compelling narrative of scientific experimentation.

Evolving Documentation: Capturing the New Research Process

In the new high-velocity development lifecycles, the documentation supporting the qualification of software development activities is becoming more challenging but also more important. Substantiating the process of experimentation needs to continue to evolve along with the tools, including highlighting the following:

- Focus on the system: Document the challenges of building and maintaining a high-velocity development system.
- Evidence of a systematic testing process: The process of experimentation is now heavily weighted toward the iterative cycles of creating or composing design alternatives, refining or selecting a design option to test, designing the test, testing, debugging, and validation, framed by the "bug as a failed experiment" paradigm.
- Highlight the human's role: Emphasize that the developer is the "dominant actor," guiding the AI and making critical design decisions based on experimental results.

Traditional metrics like lines of code are becoming less indicative of true research activity. The focus must shift to capturing the **Process of Experimentation** and the **Cognitive Effort** of the developer:

Deeper analysis of existing artifacts (Jira, Git, Pull/Merge requests). Analysis of the "content and context" of development artifacts including code, or wireframes, or ER diagrams, etc. will be needed. This includes:

- Reviewing Jira tickets now potentially having a much higher percentage of "bugs" some of which may need to be viewed as failed experiment logs
- Analyzing pull/merge requests for the technical debates that validate an approach, and
- Valuing Git commit messages for their explanation of why a change was made.

If the nature of the work is changing, then how should the methods for documenting that work evolve to capture the new process of experimentation?

Capturing the cognitive process of experimentation. To demonstrate the human-led discovery process, new forms of documentation might require consideration. For instance, an architectural decision record

(ADR) could document not just the final choice, but also the AI-generated alternatives that were discarded and the technical reasons why. A relevant example could be an ADR for a caching strategy that states: “Evaluated three AI-suggested approaches. Centralized Redis was discarded due to simulated bottleneck concerns. Client-side replication was chosen, though it requires a custom test harness to validate its logic.” This captures the evaluation of alternatives.

Highlighting the role of testing and bug-fixing. If a bug is a failed experiment, then its documentation should be treated with the same rigor. One could ask if a bug ticket in Jira could be reframed as a failed experiment log. For example, instead of a title like “Bug: Page crashes,” the new title could be “Failed Experiment: AI-generated concurrency model causes deadlock.” The description would then detail the initial hypothesis (the AI code would work), the method of discovery (the load test that caused the failure), and the resulting technical uncertainty that must now be resolved. This turns a simple bug report into a rich narrative of scientific inquiry.

Re-contextualizing existing artifacts. Even new forms of documentation, such as prompt iteration logs and ADRs are needed to make the developer’s thought process visible and auditable.

Adopt traditional artifacts like pull or merge requests can be viewed through this new lens. The discussion within a pull request may be a critical indicator of research, as it’s where the collaborative, human-led review and validation of an AI’s output occurs. A comment such as, “The AI used a recursive pattern here, which is clever, but have we tested for stack overflow with deeply nested data structures before we merge?” is strong evidence of due diligence and the identification of new uncertainties.

Considering new metrics indicative of cognitive effort. Should we consider adopting new metrics that reflect this experimental process? Instead of just lines of code, perhaps tracking the code churn rate within a developer’s feature branch could be more indicative. A high churn rate may suggest that the developer is actively experimenting, iterating with the AI, and discarding failed approaches rather than passively accepting the first output. Similarly, a rising ratio of test code to production code could signal a greater focus on the qualified activity of validating complex, AI-generated systems.

Takeaways

The shift to Gen AI-assisted development actively redefines the human developer’s role—transforming them from a straightforward coder into a systems architect and lead scientific investigator. Developers now direct experiments, validate outcomes, and engineer resilient, scalable systems that drive high-velocity innovation.

Today, the core of research activity is not simply coding, but strategically designing, testing, and managing these sophisticated R&D processes. By adopting advanced testing methodologies and rigorously documenting each step—treating every bug as a discovery and every test as an experiment—organizations can clearly demonstrate the human ingenuity and experimental rigor behind their AI-driven breakthroughs.

Practitioners can help companies operationalize these recommendations by advising on implementing robust documentation practices, establishing metrics that capture cognitive effort, and aligning development processes with the evolving requirements of the R&D tax credit. By partnering with practitioners, organizations can confidently substantiate their claims and help ensure that their investment in Gen AI innovation receives the recognition and support it deserves.