
WHITE PAPER

# Achieve Next-Level Functional Testing with Semantic AI Validation

By Yoram Mizrachi | CTO, Testing, Perforce

# Introduction: The New Era of Automated Functional Testing

Functional testing has always been a critical component of the software development lifecycle, acting as the gatekeeper to ensure applications perform as expected. At its core, functional testing revolves around validations—also referred to as assertions or checkpoints—which compare an application's actual behavior against expected outcomes.

Functional testing often compares "expected vs. actual" results, such as verifying whether the login button appears on a screen or determining if the account balance exceeds $100. However, limitations in current validation methods prevent fully automated testing from delivering the same nuanced, context-aware insights that manual testers inherently possess.

Enter Semantic AI Validation—a revolutionary approach poised to reshape the functional testing landscape. By leveraging cutting-edge technologies such as neural networks, Large Language Models (LLMs), and Vision Language Models (VLMs), semantic AI brings a contextual, human-like intelligence to automated testing.

This white paper explores the current state of validation, highlights common pain points with traditional methods, and demonstrates how semantic AI offers a scalable, precise alternative for addressing the limitations of existing tools.

# Validation Methods in Functional Testing

Modern functional testing relies on various types of validations to assess software quality comprehensively. Below, we explore some key validation approaches and their strengths and shortcomings.

## Element-Based Validations

**Element Existence**

The simplest form of validation involves checking whether a GUI element exists. For example, in Selenium, a typical code snippet like obj = driver.findElement("//xyz") checks for the presence of an element.

While it is the most common type of validation, the primary problem with "element presence" lies in how to address the element. How does a tester refer to an element on the screen using the screen object tree/DOM representation? The more modern the application is, the more complex this becomes. A typical browser or mobile application can have thousands of elements arranged in a dynamic, multi-layer approach, so identifying a specific element correctly becomes a major obstacle and a likely source of difficulties when it comes to script creation.

## Element Status and Attributes

Once an element is identified, additional validations can determine its visibility (isVisible), state (isEnabled), or other attributes such as color, font, or location.

However, these validations may not always reflect the user experience. For example, verifying a specific HEX color value might result in a "false negative" because a designer slightly altered the hue, which is insignificant to users but flagged as an issue in the test. Additionally, modern applications often combine multiple elements into complex layers to replicate a single behavior. For example, a button's "enabled/disabled" state may depend on combinations of visual cues not directly tied to a single element's state.

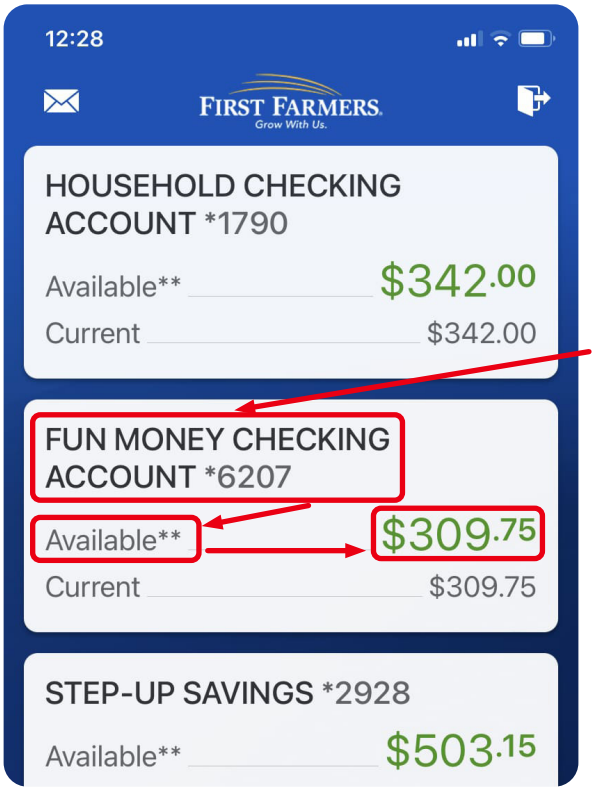
Checking the attributes of an element may be correct from a coding perspective, however, this method can overlook the user experience angle of testing.

## Element Textual Values

Advanced validations involve checking the value of a given element. These comparisons range from simple text matches (e.g., actual == expected) to more complex scenarios like "starts with," "contains," "greater than," or "between."

Code-based tools like Selenium excel here, offering unlimited flexibility for developers. However, codeless tools may lag in providing sophisticated options for validating textual values.

A common problem with value comparison is value normalization before comparison. For example, dates and currency can be represented in multiple ways (DATES: August 3rd, 03/08, 08/03/2025, etc. / CURRENCY: $23,444.34, -$234, $/€/¥, etc.). Support for normalization of such items is not easily accessible.




Source: https://www.myfirstfarmers.com/wp-content/uploads/GS-Step3-MainScreen.jpeg

# Relationships and Anchors

Validating relationships between elements and using anchors is one of the most challenging aspects of functional testing. For example:

To verify the balance of an account ending with "6207" as "$309.75," testers must perform these steps:

1. Identify all account entries.
2. Pinpoint the card matching "6207."
3. Find the "available" text within that card.
4. Retrieve the sibling object showing the balance.

This complexity often calls for a mix of logic and creativity.

In the above example of a screen semantic, the information shown has an inherent visual relationship which is bound by semantic visual meaning. The object tree/DOM represents the code-level hierarchy, which does not normally translate to visual semantics. This gap needs to be solved through complex and normally imperfect scripting and assumptions.

Every anchor along the path is a breakable locator. Shortcuts are possible, but they assume an application state which may or may not remain intact over multiple executions.

# Visual Validations

Visual validation considers what appears on the screen, making it both powerful and limited by its simplicity. It evaluates the application's presentation layer, which directly interacts with the user.

## Types of Visual Validations


- **Pixel Matching:** Comparing two screenshots at a pixel level. This approach is largely impractical due to rendering algorithms and artifacts arising from minor screen variations.
- **Pixel Similarity:** Analyzing images as pixel arrays, with algorithms accommodating variations like scaling, rotation, and minor rendering differences.
- **Template Matching:** Unlike pixel similarity, this method identifies a source element within a larger target image. It works well for locating specific icons or controls.

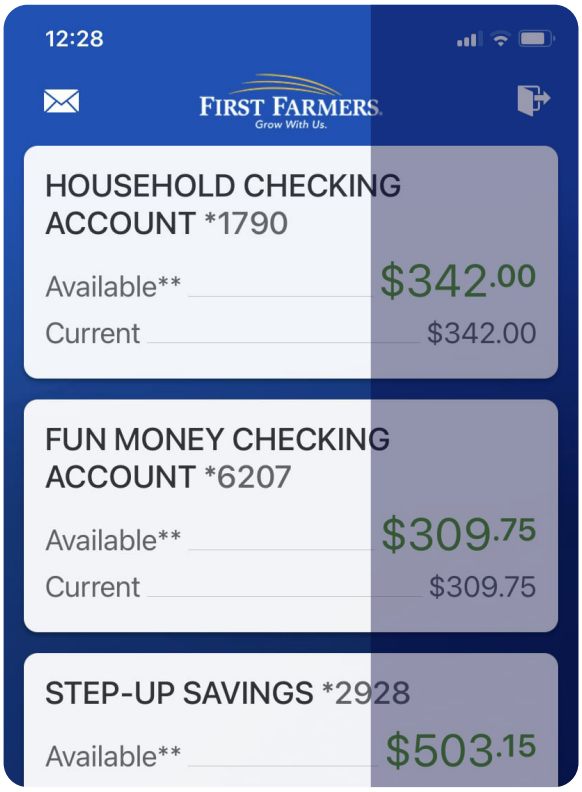
Unless you are an expert in CV (Computer Vision), do not attempt to use simple pixel-level comparisons for your visual validations, as they often do not work as expected.

Visual validations use many different terms and algorithms from the CV (Computer Vision) space, many of them involving Neural Network or Machine Learning algorithms, which can be categorized under "AI." However, it is important to define AI differently in this instance, based on the models and practices used. In other words, not all AI is created equal.

## Practical Implementations of Visual Validation

### 1. Find Image Matching

This approach, akin to finding a needle in a haystack, identifies icons or specific features in a broader visual context.




Source: https://www.myfirstfarmers.com/wp-content/uploads/GS-Step3-MainScreen.jpeg

### 2. Selective Exclusion Matching

By excluding irrelevant parts of the screen (like clock displays or reception bars), testers can focus on validating key structures within the application under test.

> The biggest issue with "selective exclusion" lies in the practical use of it—the most overlooked areas or elements are more likely to get a "false positive" result, whereas the most observed areas are more likely to get a "false negative" result. In other words, selective exclusion is prone to human error.

## Optical Character Recognition (OCR)

OCR leverages machine learning (ML) and neural network (NN) algorithms to extract text from visual interfaces. It excels at recognizing isolated text, such as "$309.75" appearing somewhere on the screen.

While OCR serves basic text-detection needs, challenges arise when precise spatial relationships between text (e.g., "$309.75" next to "6207") matter. OCR engines may also misinterpret similar-looking characters, requiring custom script corrections to minimize errors using normalization techniques (e.g., Levenshtein Distance).

> While visual validations shine in their simplicity for end users, they have hard limitations when it comes to context and semantic understanding. Just like object-based validations, with visual validations it is up to the script writer or tester to manually create the context and relationships.

# Media Validations

For audio and video applications, validations extend to tasks like audio matching, speech-to-text analysis, and lip-sync verification. These processes rely on AI-powered tools to deliver quality assessments and ensure seamless playback experiences.

**Audio matching** – Ranges from simple waveform/frequency matching (similar to pixel matching in images) to template matching (like song recognition).

**Speech to text** – When required to work with values (for example, in IVR testing), speech-to-text is a useful approach to translate a waveform audio stream into a textual representation. This textual output can then be used in a scripting environment, leveraging existing "value matching" techniques. Similar to OCR, speech-to-text also employs ML/NN algorithms and can, therefore, be categorized as AI.

**Audio quality** – Unlike matching, quality validation involves different algorithms (such as PESQ, MOS, or POLQA) that assess the (often subjective) quality of the audio. This is typically used to ensure that streaming quality meets the expected level.

**Video quality** – Similar to audio quality validation, video quality assessment examines both visual and audible aspects, as well as the synchronization between them (e.g., lip-sync testing).

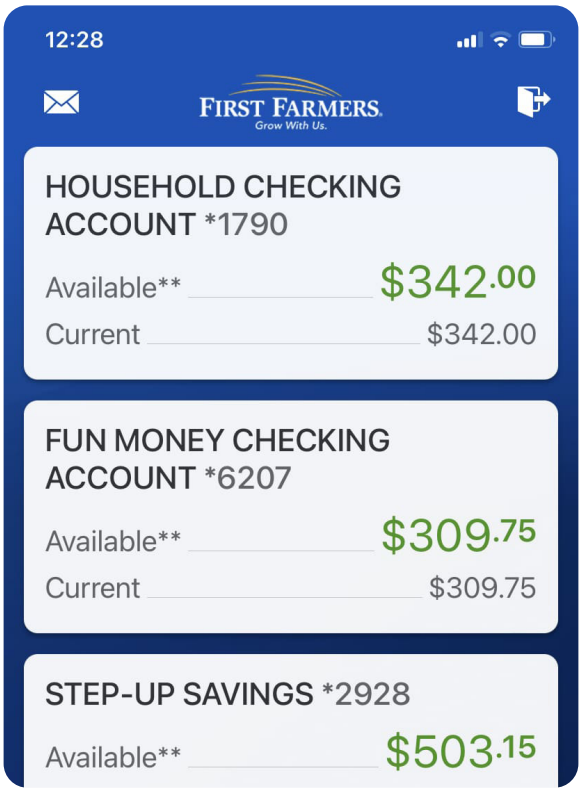
# Semantic Validations

## Understanding Semantic Meaning

Semantic meaning stems from the relationships between objects, interfaces, and context. For example:

- **Linguistic Semantics** involves syntax, vocabulary, and contextual relations.
- **Visual Semantics** applies to UI elements like layouts, colors, and interactions, illustrating meaning without textual reliance.

Semantic validation bridges these domains, interpreting both textual and visual inputs to understand relationships, intents, and contextual nuances.




Source: https://www.myfirstfarmers.com/wp-content/uploads/GS-Step3-MainScreen.jpeg

## Semantic Validation in Manual Testing

Manual testers naturally interpret semantic contexts. Using the above example, "The available balance for the account ending in 6207 should be more than $100," a human can:

- Recognize cards representing accounts.
- Identify the matching account using "6207."
- Pinpoint the associated "available" balance ($309.75) and confirm it exceeds $100.

Even flexible, idiomatic inputs like "fun money should be more than a c-note" are interpreted correctly in manual testing. Translating such semantic understanding into automated solutions, however, is complex.

## Automating Semantic Validations

Semantic validation powered by AI bypasses traditional limitations while retaining the nuanced capabilities of human interpretation. It relies on large language models (LLMs) and vision-language models (VLMs) that merge textual and visual inputs into unified semantic frameworks.

To ensure accuracy, semantic validation incorporates:

- **Structure:** Clear outputs (e.g., PASS/FAIL) based on expected and actual results.
- **Robustness:** Consistency between test executions.
- **Training:** Clear definitions to guide validation logic.
- **Timeline Awareness:** Context-sensitive questioning to track state changes over time.

> Without Structure, Robustness, Training, or Timing, it will be very difficult to integrate an off-the-shelf model into your automated testing workflow.

# The Human Factor of AI Semantic Validation

The cornerstone of successful Automated AI Semantic Validation lies in the validations themselves. Simply put, if you do not ask the right questions, you will not get the right answers.

Consider this: if your validation states, "the application should look nice," the result is inherently subjective, hinging on countless factors that define "nice." Or take, "the account balance for fun should be sufficient"—what does "sufficient" even mean? It is open to interpretation. Ambiguity can also creep in with validations like, "the button should be green," when multiple buttons are present, or with questions that lack clear PASS/FAIL criteria, such as, "what is the balance of the account ending with 6713?"

While any textual input might seem valid, without a proactive, assistive framework to keep you on track, it is all too easy to create vague or incorrect validations. The key to success? Precision, clarity, and the right guidance to ensure your validations deliver actionable insights.

# AI Validation

Perfecto's AI Validation transforms the way teams approach testing by intelligently analyzing test scenarios, visually validating applications, and adapting to changes—all without manual intervention.

- **Simplify Testing with Plain Language:** Enable your team to contribute to testing with simple, plain-language scripts—eliminating complex scripting.
- **Comprehensive Validations for Better Coverage:** Validate functional, visual, and contextual elements to ensure improved coverage across all application layers.
- **Reduce Maintenance, Focus on Strategy:** Automate script creation and updates to eliminate repetitive tasks and enable your team to prioritize high-value strategic initiatives.
- **Maximize Testing ROI:** Increase efficiency, reduce manual effort, and deliver tangible value with advanced AI-driven solutions that optimize testing processes.

Learn More About Perforce's AI-Powered Testing Tools

# Final Thoughts on Semantic AI Validation

For decades, manual testing has dominated functional software testing, offering robust and context-aware validations at the cost of scalability, speed, and fatigue. Meanwhile, traditional automated testing has struggled with limited, rigid validation models ill-suited for modern testing needs.

AI-driven semantic validation introduces a future where testing benefits from automation's efficiency without losing the context-rich depth of manual testing. By leveraging powerful neural networks capable of monolithic semantic understanding, this approach promises to revolutionize software quality assurance, ensuring applications are functional, user-friendly, and resilient in today's complex, fast-paced environment.

## See AI Testing in Action— Request Your Custom AI Demo Today

**Get Custom AI Demo ▶**

blazemeter.com/ai-testing-demo-request

# About Perforce

Perforce powers innovation at unrivaled scale. With a portfolio of scalable DevOps solutions, we help modern enterprises overcome complex product development challenges by improving productivity, visibility, and security throughout the product lifecycle.

Our portfolio includes solutions for Agile planning & ALM, API management, automated mobile & web testing, embeddable analytics, open source support, repository management, static & dynamic code analysis, version control, and more. With over 9,000 customers, Perforce is trusted by the world's leading brands, including NVIDIA, Pixar, Scania, Ubisoft, and VMware. For more information, visit perforce.com.

**Contact Us ▶**

perforce.com/support

### Related Resources:

- [eBook] The Future Is Now: Mobile & Web Application Testing With AI
- [Report] The State of Continuous Testing Report
- [Blog] The Current State & Future Trends of AI in Software Testing
- [Blog] Perfecto by Perforce Announces AI Validation—A Paradigm Shift In Testing
- [Webinar] The Future of Testing: A Conversation About the Use of AI and ML
- [Webinar] Simplify, Scale & Succeed: Perfecto AI Validation In Action