Tricentis

AI-powered test automation for the next generation of applications

Tricentis

Table of contents

Tricentis

AI-powered test automation for the next generation of applications

EXECUTIVE SUMMARY

Tricentis Tosca, powered by Vision AI is a new, next-generation technology using advanced machine learning and neural networks that elevates test automation to a new business discipline. Unlike first generation script-based technologies which produced fragile and costly maintenance, Tosca's Vision AI can be combined with model-based test automation to deliver faster, more resilient test automation across new tiers of technology.

Core business benefits of Vision AI together with model-based test automation are:

- **Lower cost of maintenance**
- **Speed to market**
- **Reduced risk**
- **Release predictability**

This whitepaper describes how Vision AI sees and steers elements on virtually any technology – from cloud-native, enterprise apps, to simple designs and mockups. This allows tests to be maintained with zero code in a human-readable model that anyone can understand and adopt. As such, Tosca eliminates the maintenance issues inherent in traditional testing methods, allowing organizations to deliver blazing-fast software innovation across their entire digital landscape.

WHY AI-POWERED TEST AUTOMATION?

In the year 2018, Tricentis developed the hypothesis that if we are to truly shift testing to the left – that is, build automation before the UI is created, as well as create resilient automation for modern applications that get updated daily, it will require a dynamic approach that entirely separates an application's technical layer from automating it. In other words, only if we completely remove the need for technical identifiers (such as Class, ID, XPath etc.) and replace it with visual identifiers driven by AI-powered automation, will it truly deliver resilient test automation that keeps up with perpetual technological change.

Why did Tricentis take a new approach by adding AI-powered test automation and build it on top of model-based test automation approach?

AI-powered test automation is our response to the rapid rise of modern applications that are built on new tiers of technology from cloud native applications, microservices to containers. These new applications get updated daily, and script-based testing approaches have been unable to keep up.

At the same time, what we witness in our customer's portfolios is a mixture of modern and legacy systems that are often integrated in complex ways. A small change in one system can have rippling effects in other integrated technologies. According to MuleSoft (2020), the average organization uses more than 900 applications today. And a single transaction can touch more than 82 types of technologies. The terrible truth is that very few companies have real end-to-end tests, and those that do – the tests are not running when they are most needed.

UI automation also continues to be a bottleneck in quicker testing, and agile delivery. The vision of Behaviour Driven Development (BDD) was to enable automation to be written by the business prior to the product existing. This was a struggle due to the fact that the automation is only ever written after the User Interface (UI) of an application is built. As a result, UI automation continues to be slow, and being slow is costly. The sooner you find a bug, the cheaper it is to fix it. This is why it is essential to test as early as possible in the development lifecycle – before the UI is created.

To keep up with an ever-changing application landscape, and enable organizations to cope with their modernization initiatives, at Tricentis, we present Tricentis Tosca powered by Vision AI, the next evolution in test automation. Vision AI is a patented deep learning technology that delivers stable, self-healing, platform agnostic UI automation.

Together with Tosca's model-based test automation and Vision AI, organizations can reduce modernization costs and efforts, as well as build resilient end-to-end testing that extends across platforms and technologies.

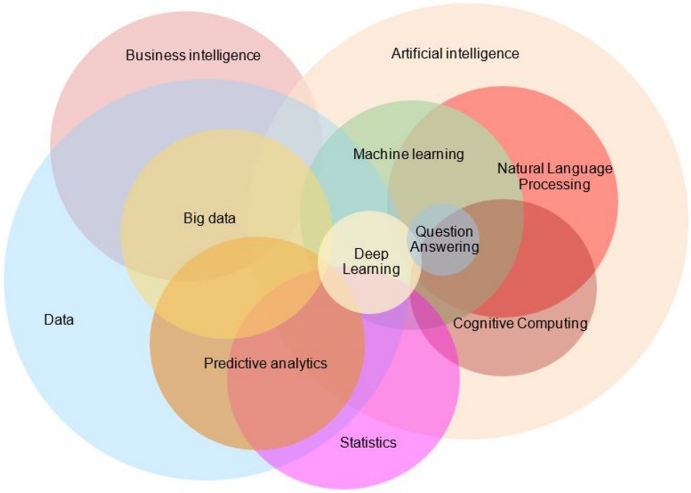
AI AND MACHINE LEARNING IN CONTEXT: WHAT ARE NEURAL NETWORKS?

Artificial intelligence (AI) as a concept is ubiquitous in technology, business, science, medicine, economics, and many more other fields. The word "AI" is so commonplace nowadays that readers are usually expected to understand from context what is meant.

However, there are common misconceptions on AI and machine learning (ML) including algorithms, as well as ways in which ML technologies learn over time. These concepts have implications on what the AI technology can claim to do and which future challenges it can solve. Let's clear some air and define some of these concepts below.

Artificial Intelligence vs. Machine Learning

Think of artificial intelligence (AI) as the umbrella term and machine learning (ML) as a sub-component of AI. Broadly speaking, artificial intelligence is anything that can perform intelligent tasks that would normally require a human to do. This loose definition makes AI applicable to all sorts of areas like big data analysis, predictive analysis, SQL queries analysis and more.



Machine learning, on the other hand, is a system that "continuously learns" and "improves" over time given more "experience". This is quite an important differentiator as over time, machine learning systems will become more future proof.

Consider the following example: your task is to find the fastest route from A to B.

A machine learning system would know the problem (A) and the result (B), but not the part in the middle, that is, how to get there. The part in the middle is what you are expecting the ML system to "learn".

What is the difference between ML and an algorithm?

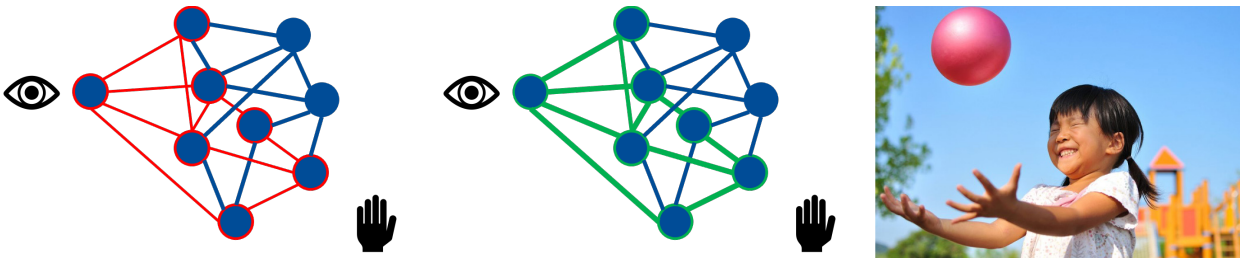
An "algorithm" works differently to ML as it involves expertise and knowledge being added into a system in order to solve a particular problem. In the example above, an expert would use an algorithm to instruct the system how to get from A to B. The system is therefore only as smart as the knowledge or expertise that we build into it.

What are neural networks?

Figuratively speaking, a neural network works like a randomly configured "brain". It is composed of an interconnected web of neurons which talk to each other. Similar to the brain – as you grow, learn and expose yourself to more experience, the chemistry of the brain changes. Certain pathways reinforce and certain pathways die off. The goal of neural networks, therefore, is to mimic the brain – or as IBM coined it in 2001, model the behavior of a computing system against the human autonomic nervous system (ANS).[2]

How does a neural network learn?

Consider the task of catching a ball. We get an input from our eyes and the motor neural response to our hand. With time, you learn to catch the ball more effectively because the pathways in between your neurons are either reinforced or weakened based on the feedback of the result. Artificial neural networks work in the same way. We train the system by feedbacking it with a set of problems and solutions, and punishing or rewarding the system if the response is correct or wrong.



[2] In 2001, IBM coined the term „autonomic computing" and modelled their approach against the human autonomic nervous system (ANS). The ANS regulates critical bodily functions such as homeostasis without conscious involvement. Applying this idea to computing, systems should adapt to their environment through self-configuration, self-optimization, self-protection and self-healing. References from: https://www.aitesting.org/

To train a machine learning system to correctly deal with problems or tasks, we need to feed it with lots of training data. Training data is usually a large set of examples, and these should include:

- Examples of good results
- Examples of bad results
- The ability to update our system

Have a look at these images:

It's not rocket science to figure out that these are images of airplanes, despite seeing some of them for the first time. Simiarly, we train the system to figure out that this is an airplane because we feed it with large amounts of human labelled data and punish or reward the system based on the output given. Labelling data is the process of naming each individual image with a label such as "text_field", "search_ button", "scroll_down_menu".

Finally, all of this labelled training data is passed onto a machine learning system and we say to that system to 'train'. The ML system starts randomly guessing the correct label for each object. If it is correct, it remembers how the pathways were modelled. When it is wrong, it remodels those internal pathways and tries again. This process happens thousands of times until it can correctly identify most labels for objects.

What are the different ways in which a machine learning system learns?

ML can learn through various techniques, like supervised, unsupervised or reinforced learning. The process we explained above is referred to as **supervised learning**. This involves feeding the system with many examples, rewarding and punishing the system based on the output given until it eventually reproduces the correct output. The downside of this approach is that you need accurately labeled data, which is a labor-intensive process.

In **unsupervised learning**, you do not require labelled data. You simply expose your model to data and let it find interesting patterns that could be relevant. Some use cases include Google News which categorizes articles according to the same story from various online news outlets. For example, news about the COVID-19 vaccine would be categorized under that appropriate label. Unsupervised learning techniques could be less accurate than supervised learning. However, they don't require human intervention to label the data appropriately. The use cases are therefore slightly different.

Finally, there's several other ML techniques like adversarial learning and reinforcement learning, which we won't get into due to the scope of this paper.

How can AI/ML be applied to software testing?

A primary objective of test automation is to be able to identify objects on an application in order to perform a set of actions with those objects and arrive at a desired outcome. This could be clicking on a button, verifying values, scrolling through a table, searching for a data record, selecting a value from a drop down. As humans, we readily perform those actions just by seeing the objects on the screen. Our eyes see the "enter" button and we click it if we want to proceed to the next step of, for example, our online shopping process.

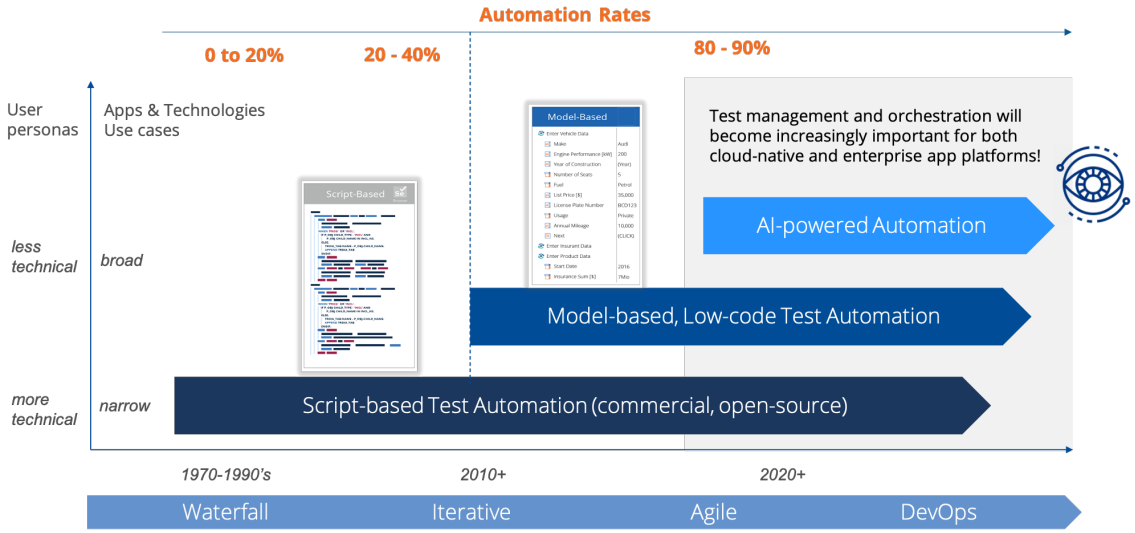
As humans we are also particularly good at recognizing the variety of objects that exist – we know that

next to a field called 'Name' there is a text box, a magnifying glass is a search bar, a logo is a picture and 'Read more' in blue is a link. What's more, we don't care whether those buttons or objects are designed with a different color, font or size, or are positioned differently on a page. As humans, we are good at intuitively and easily identifying those different types of objects or controls.

Much like humans have the ability to interpret various application screens and objects, we can teach machines to do the same.

THREE GENERATIONS OF TEST AUTOMATION

Test automation has drastically evolved over the years – from script-based testing and test automation frameworks to model-based testing. The purpose of this section is not to provide you with a full-blown historical overview of how test automation has matured, but to highlight that the same test automation frameworks that were used over a decade ago are still being used today in the wake of modern DevOps – and agile teams can't keep up. As application development architectures evolve in DevOps with cloud-native, Kubernetes, infrastructure as code and microservices, test automation also needs to evolve, too – urgently.



First generation: script-based test automation

The problem with script-based test automation is that the automation is rigidly integrated with the code of the application under test, and this makes the test cases very fragile under change.

The first generation of test automation was **script-based**. Script-based testing uses **record-and-playback** solution where a tester records their UI actions and replays them to test when needed. Unfortunately, script-based tests are very fragile. Any significant changes to the application require considerable programming expertise to run. Other challenges include:

- Test data and test logic is hardcoded into the test script
- Applications readily change, requiring expensive resources to maintain the test scripts
- No use of synchronization like waiting for a window to appear
- No way to reuse code snippets
- Not possible to automate across various business applications and systems
- Low automation rates of 0 – 20%

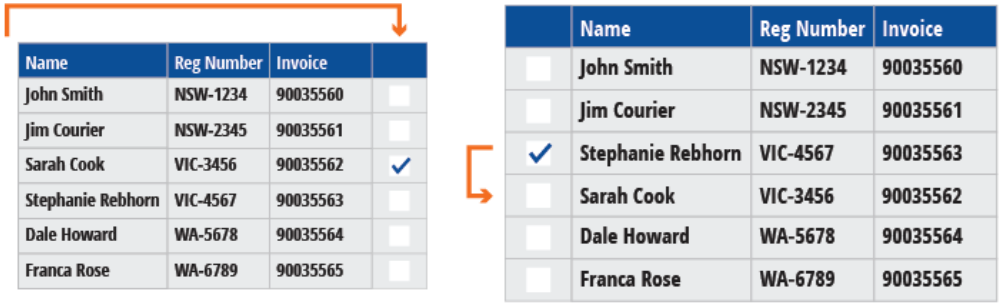
Consider the example below. The task is to select the checkbox for Sarah Cook. The translated code for this action simply states "**click a checkbox in column #1 and row #3**".

	Name	Reg Number	Invoice
☐	John Smith	NSW-1234	90035560
☐	Jim Courier	NSW-2345	90035561
☑	Sarah Cook	VIC-3456	90035562
☐	Stephanie Rebhorn	VIC-4567	90035563
☐	Dale Howard	WA-5678	90035564
☐	Franca Rose	WA-6789	90035565

→ `Checkbox Click, "/usr/cntlCONTAINER/shellcont/shell[2]/chbx[1,3]"`

However, what would happen if:

a. you switched the columns around, i.e. you put the 'checkout' column as the last column, and;
b. you switched the rows, i.e. you put John Cook beneath Stephanie Rebhorn?

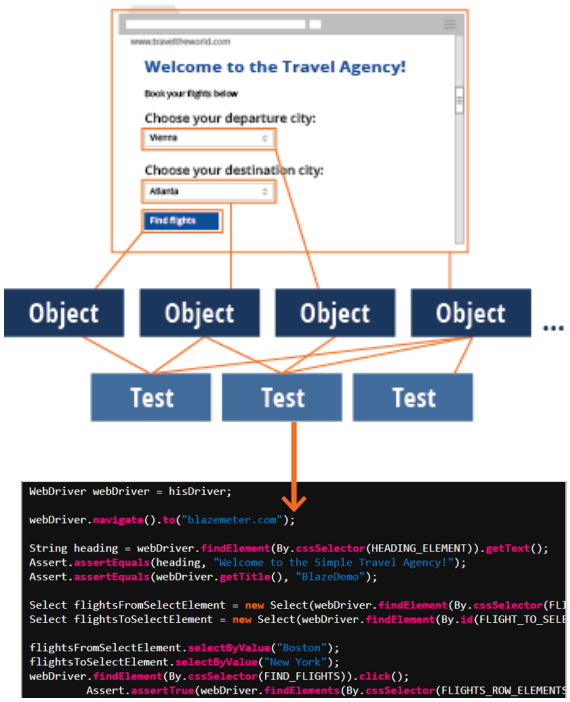


The test case would break on both occasions:

a. The automation code will not be able to locate any checkboxes in column #1 that can be clicked and will therefore fail to execute.
b. John Cook now appears in row #4 instead of row #3. The code will execute without technical failure but it will not select the invoice for John Cook as originally intended. The objective of the test won't be met, and thus result in a semantic failure of the automation code.

The first generation of test automation also saw the emergence of **test automation frameworks**. These include commercial and open-source tools. With frameworks, the automation scripts are broken down into components or snippets of code. These snippets of code can be used and placed into libraries for more reusability. Unfortunately, frameworks suffered from the same problem as record/playback solutions. Each test step is written in procedural code such as Python or Java, and is hard-coded, step-by-step, to interact with the objects in the application. This renders the tests very fragile under application change and is costly to maintain.

Consider the example below. If one object in an application changes, you have to update each and every code snippet that relate to that changed object. Additionally, there could be hundreds if not thousands of code snippets related to a changed object. On top of that, this approach leads to less-than-optimal automation rates of only 20-40%.



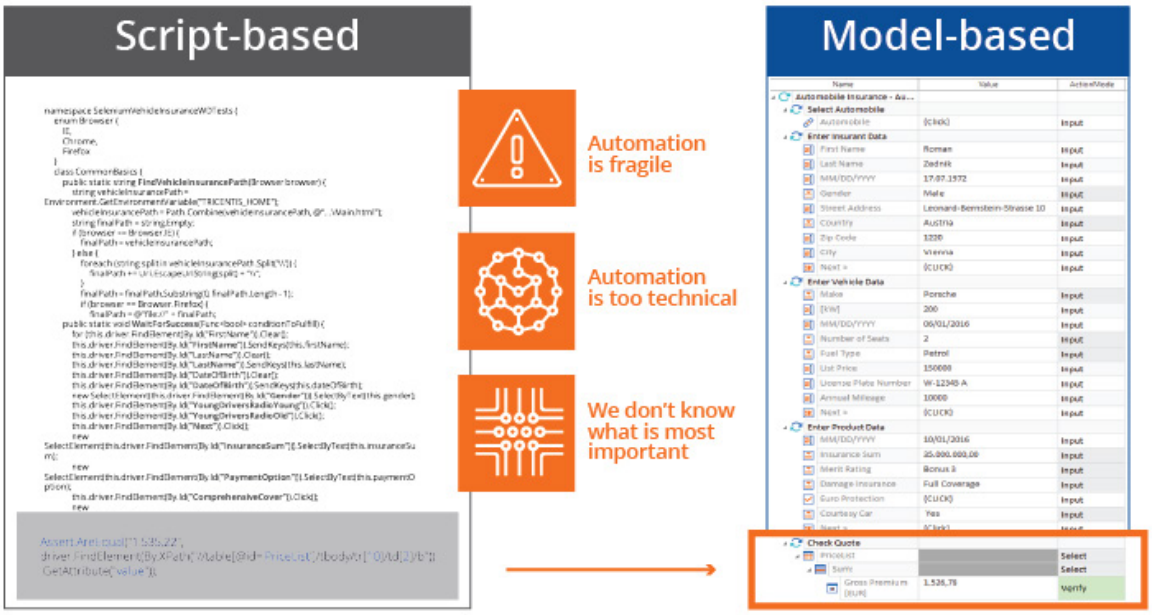
Second generation: Model-based test automation

Model-based test automation decouples the automation model from the technical layer of an application. This renders test case creation and maintenance much faster and less error prone. Test cases are maintained in a codeless, human-readable model that does not require technical resources to build. The combined result is significantly reduced maintenance and high automation rates of 90% and higher.

Instead of programming a test automation framework, with model-based testing you rapidly scan the application under test to create a business-readable automation model through a no-code approach.

The automation models are called modules in Tosca. When a change occurs in the application under test, e.g., an object or attribute name changes, you simply update the module and that change will automatically be propagated to all your impacted test cases, making it fast and easy to maintain.

Model-based test automation empowers anyone from non-technical people to business experts to contribute to test automation.



The core benefits of model-based test automation are:

- **Significantly reduced maintenance –** there is a clear separation between the automation model (e.g. steering) and the technical details, such as the test data and test logic. The test logic and test data are injected into the automation model at runtime—guaranteeing that tests never use old versions of test data or access outdated technical definitions. Tricentis customers achieve repeatable 90%+ automation rates – read some of our customer success stories for more information.

- **Anyone can use it –** as there is no coding or scripting knowledge required to build the automation, model-based testing makes it suitable for multi-skilled personas, from business experts, to manual testers, non-technical users and other SMEs. Customers have been able to cut costs on technical resources and achieve automation across teams.

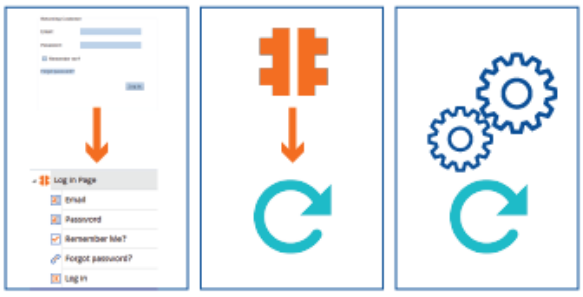
- **Maximize reusability –** build a module once and reuse it across several test cases. There are also several features within Tosca such as libraries, reusable test step blocks and standard modules to increase the speed of your test case creation and maintenance.

- **Scriptless test automation for over 160+ technologies:** you can test virtually any technology from UI applications (web, mobile, or custom), API, programming languages, mainframe, databases, to critical enterprise applications like SAP, Salesforce and ServiceNow. Tosca's extensive technology support enables organizations to achieve true enterprise end-to-end testing across all layers of enterprise architectures and business applications.

In a nutshell: how Tricentis Tosca works

Simply scan the application's interface (UI or API) to create the modules. Drag and drop these modules into your test case. Add values to your test case, such as the test data and test logic. Instruct the test steps on what action they need to perform, such as verify, input, wait on or buffer (which means save a value for later validation). Finally, drag the test cases onto execution lists and run them when you are ready.

Check out the Tricentis Academy for courses on Tosca as well as endless library of resources and interactive videos – free for your learning needs.



AI-powered test automation for the next-gen applications

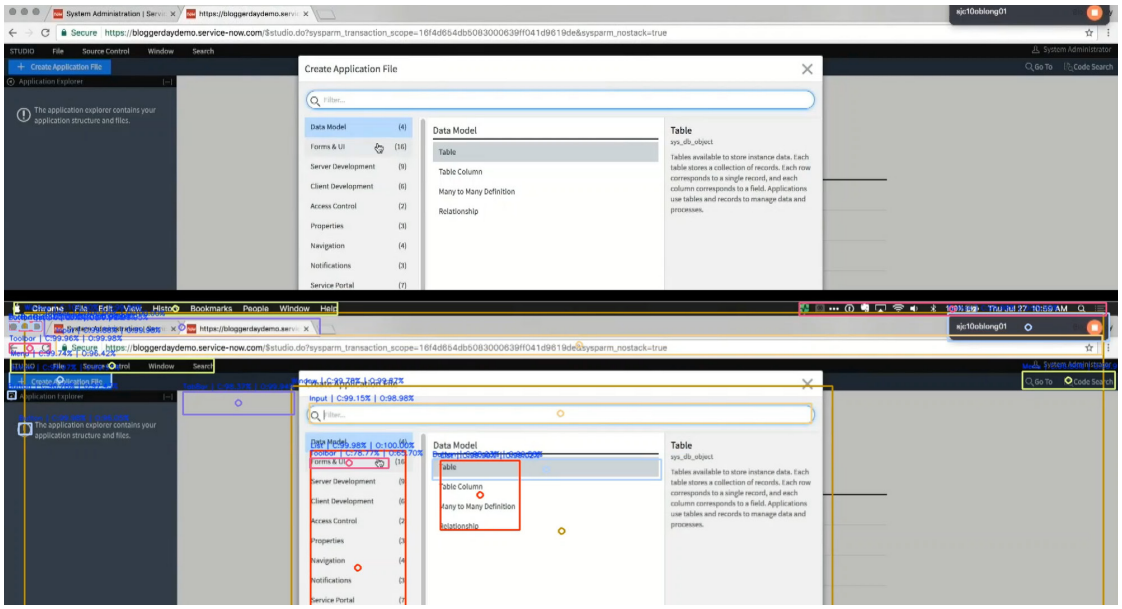
With the launch of Tosca 14, Tricentis delivers Vision AI, the next-generation AI-driven test automation technology. Vision AI allows you to automate UI-based test cases, including cloud-native applications, remote desktop applications and even design mockups before any code is written – enabling you to test much earlier in the development lifecycle.

Vision AI integrates seamlessly into the powerful Tosca test automation suite and enhances the current model-based testing approach, enabling you to extend your end-to-end testing capabilities across multiple technologies.

Vision AI: advanced neural network technology

Tricentis Vision AI is a patented deep learning technology that works by using convolutional neural networks, combined with advanced heuristics to deliver stable, self-healing, platform agnostic UI automation. Vision AI removes the need of looking at the code underlying the technology such as ID, tag name, class, or XPath etc. Instead, Vision AI uses visual clues for identifying objects and controls, and steers them irrespective of the underlying technology layer.

As with other advanced machine learning systems, Vision AI improves its ability to handle more complex controls with time, as well as diverse applications, platforms and technologies. This is because the machine learning model is consistently configured and reconfigured throughout its training process as new data arrives.



What is different about Vision AI?

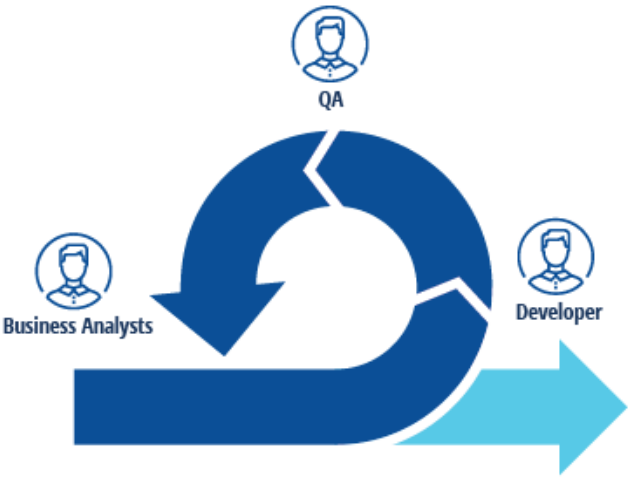
A general rule of thumb is that Vision AI can supplement engines within Tosca where automation might be difficult or impossible. Some of these areas include:

- **Simple designs and mockups**
- **Virtual or remote desktop infrastructures (VDI)**
- **Hard-to-reach interfaces**
- **Modern or legacy applications that require heavy customization**
- **Frequent upgrades and updates to applications**

Let's explore these use cases in more depth.

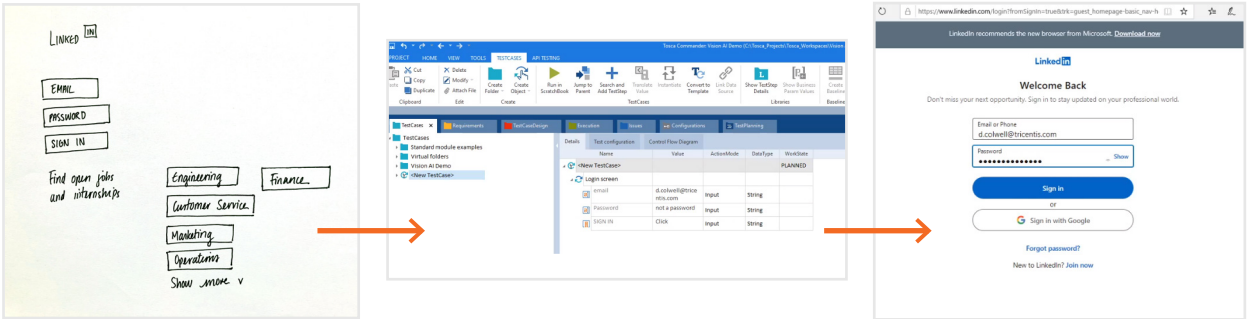
Extreme shift left – convert mockups into automated tests

In agile sprints, you normally design the mockup or simple drawing of an application in the initial phase of the software development lifecycle, before any code is written. During this phase, testers are left in a state of complete inertia – they can't begin testing until the UI is fully developed. Ultimately, this slows down testing, agile delivery and lowers the efficiency for teams striving to adopt DevOps practices.



With Vision AI, you can create test automation in the design phase of an application and run those same tests as the application evolves. Vision AI enables an extreme shift left approach to UI testing, enabling you to achieve in-sprint automation at the speed of DevOps.

Below you see an example of Vision AI scanning a simple drawing or a whiteboard, creating the modules in Tosca and running the test cases successfully for the application under test.



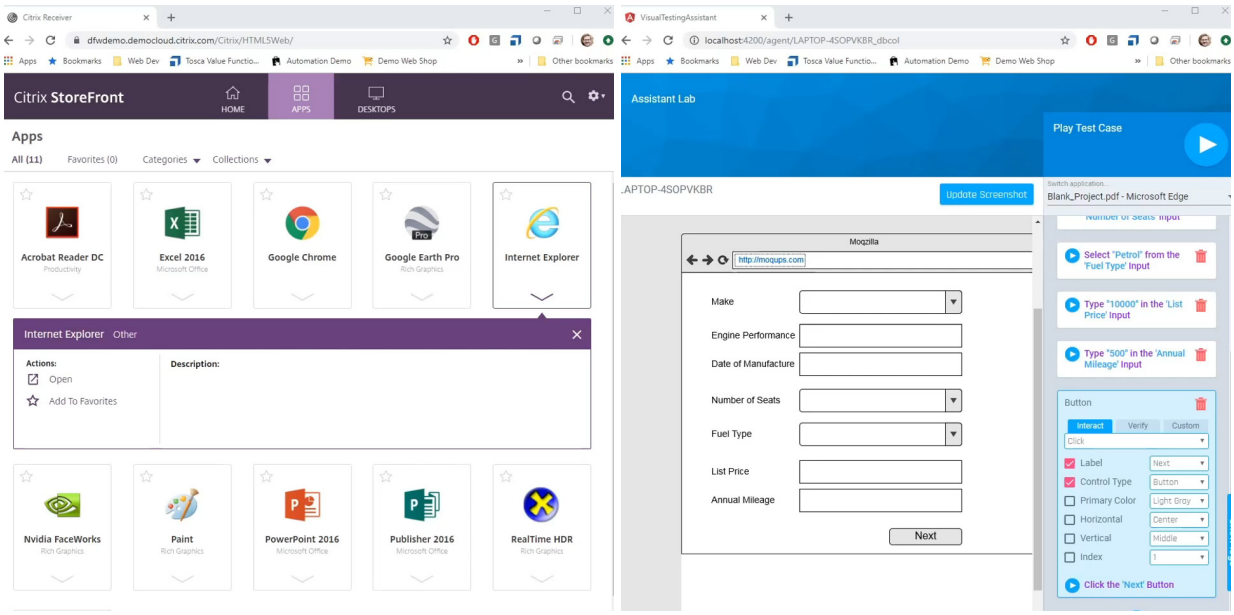
Scan **Create test cases** **Create test cases**

Testing Citrix and remote desktops infrastructures

The COVID-19 pandemic has shifted much of the world to remote work environments. This has led to an even greater demand for virtual and remote desktop interfaces like Citrix and VMware. Microsoft (2020) reported that the adoption of Windows Virtual Desktop (WVD) had tripled since the start of COVID-19.

Despite its popularity, any tester who has tried to automate tests for Citrix or other remotely hosted applications will know the pain involved. To put it simply, there are no objects to identify on Citrix – all you see is the image of the application. With Citrix, you can't access the underlying technical properties like ID and tag name, as the application is being streamed through a server. So, you must figure out another way to interact with the objects. You can automate Citrix through image-based recognition. However, even here you have the problem of creating stable tests when the streamed image has pixel differences or low screen resolution, including loading time.

Humans can readily recognize objects on a screen, regardless if it's being streamed through Citrix or any other virtual desktop infrastructure (VDI). Vision AI does the same. With Vision AI, you can create reliable Citrix automation with ease, as it detects objects on an interface by looking at the visual cues only, and not at the underlying technical layer.

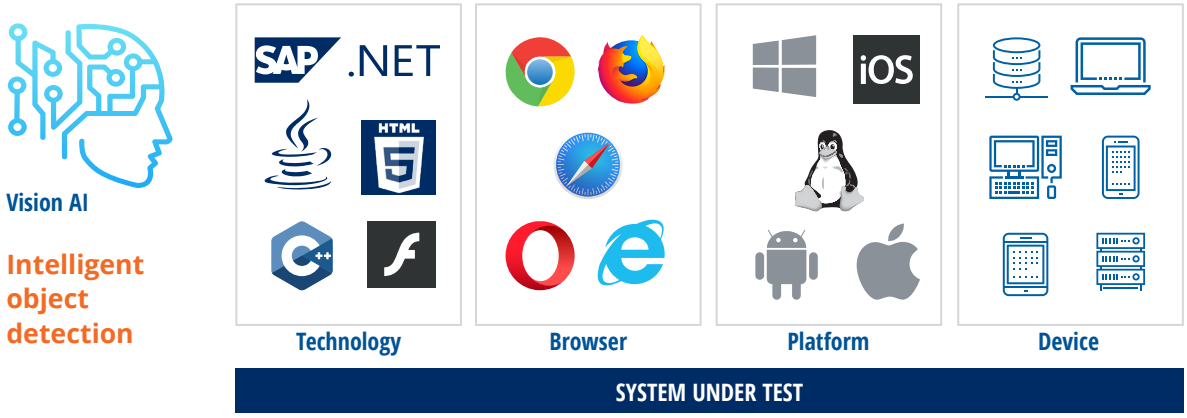


Asides from virtual environments, Vision AI is capable of automating a host of other hard-to-access digital interfaces:

- Point-of-Sales (PoS) systems used by many retail providers, which are mostly locked down environments and have no internet connection for security reasons
- Physical devices, i.e. a life science provider may manufacture high-precision machines used for automating physical processes in laboratories, like pouring liquid
- Other examples include mobile devices, emulated devices, IoT devices, infotainment systems or other non-android devices

Build once, run on any technology

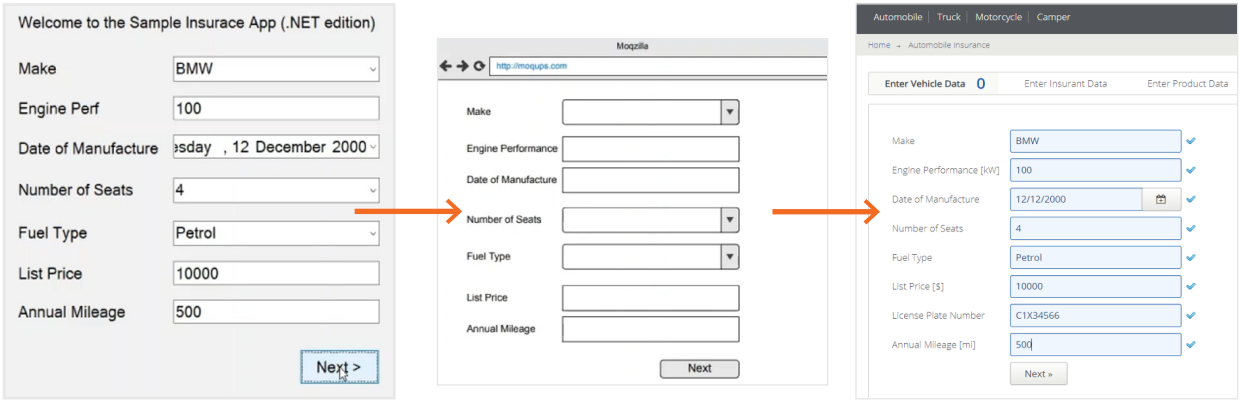
Tests that you build with Vision AI are reusable across different platforms, devices and technologies. You build it once and gain high stability every time there is a UI change. This is particularly critical for end-to-end testing, which requires a high level of reliability and scalability when testing across a range of different systems and technologies.



Keep up to date with frequent application upgrades and updates

You can't afford to spend time fixing tests with every new build; you want the same tests to run as expected as the application gets updated. Vision AI delivers flexibility and resilience while reducing the fragility that comes with script-based testing solutions – allowing you to continuously run the same tests across different builds of the same application. In other words, Vision AI is more robust and adaptable to the expected changes that inevitably occur with each application update and upgrade.

With Vision AI, tests do not need to be rewritten as the UI changes because Vision AI automatically detects controls like the human eye. Vision AI would successfully run the same test across different versions of the same application.



Modern or legacy applications requiring heavy customization

Modern application development is rising at a phenomenal rate. IDC predicts that by 2023, over 500 million new applications will be developed. At the same time, customers rely on old technologies or legacy systems for performing various business processes.

Creating automation for both new and old technologies is challenging as these applications may not be supported by most automation tools and therefore require extensive customization in order to automate them. For example, if the application is written in C++ Qt, which might not be supported by Tosca's native engines, we would build the automation via Vision AI. Vision AI can also work with new or specialized technologies such as Flutter, Blazor, and Electron. As with old technologies, Vision AI can support outdated or deprecated versions of Gupta, Silverlight, and Flash.

Other use cases involve complex controls that may not be identified out-of-the box using most automation tools. Customization is a time-consuming and costly process, and Vision AI helps to lower the time and resources needed to build automation for a plethora of old and modern tech.

Train Vision AI

You can train Vision AI to work with controls that are not recognized correctly, for example if a table is not recognized as a table. You can use the feature User Identified Controls (UIDCs). This feature enables Vision AI to recognize a control correctly using surrounding controls as anchor points. Vision AI uses these anchors as reference points to identify the control.

INTEGRATING VISION AI WITH MODEL-BASED TEST AUTOMATION: THE BEST OF BOTH WORLDS

Is Vision AI separate to Tosca?

Vision AI is not a standalone product but an AI-powered engine within Tosca version 14 and above. The Vision AI engine will not replace existing engines within Tosca – rather, it works in tandem with existing Tosca capabilities to extend test automation into many new areas and use cases. This will allow you to automate more, test much earlier in the sprint cycle, keep up more easily with application changes as well as handle complex modern and legacy technologies.

How does Vision AI work with model-based test automation?

To clarify, Vision AI does not differentiate to model-based test automation. Instead, Vision AI augments and supplements the existing Tosca approach with added ML capabilities. You can use Vision AI tests alongside other engines within Tosca. For instance, you can mix Vision AI modules together with modules created by the HTML engine into the same test case. You can also combine Vision AI test cases with all other capabilities in Tosca to build end-to-end test automation across platforms and technologies, such as:

- **Risk Based Testing**
- **Test Data Management**
- **Service Virtualization**
- **API testing**

To use Vision AI in Tosca, you do not require any technical know-how or coding knowledge. This allows business users and non-technical users to easily be embedded into the test automation process. To create Vision AI tests in Tosca, simply scan your application or system under test using the Vision AI engine, and then drag and drop your modules into your test case. The user experience is the same as creating a normal test case using XScan. The only difference is, when running your test case, your Vision AI test steps will be executed using the Vision AI engine.

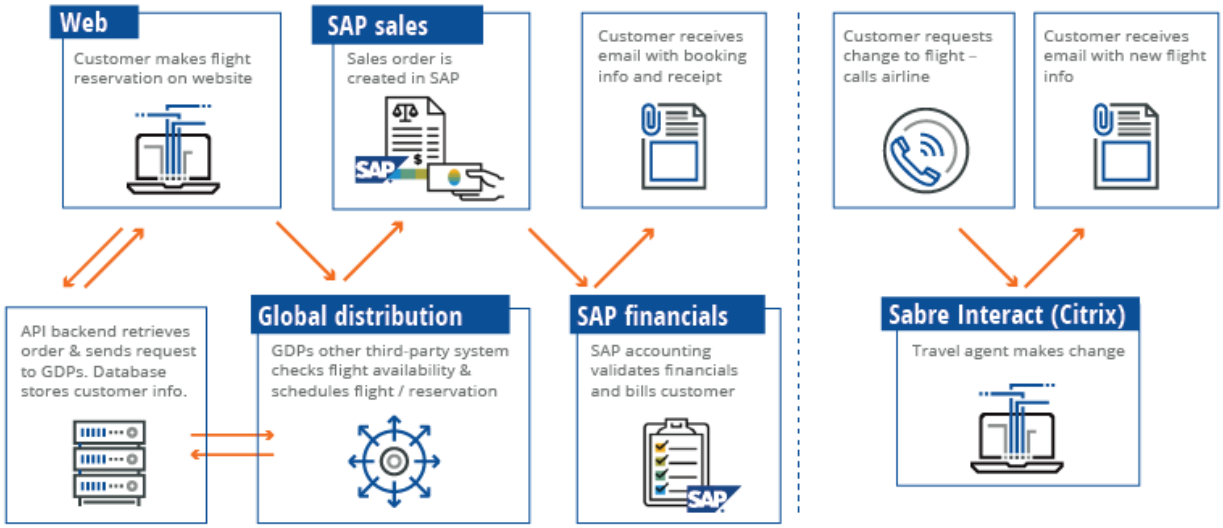
If you want to learn more about how to use Vision AI in your workflows, check out the Tricentis Academy YouTube Channel as well as the Manual.

Real-world example - Vision AI for enterprise end-to-end testing

Automating across enterprise business applications and processes using AI-powered automation is the key for success. By combining the power of model-based test automation and the cognitive capabilities

of Vision AI, organizations can achieve a whole new direction in enterprise end-to-end testing. Creating a frictionless customer experience is at the center of this process. In order to ensure that innovation for the customer can happen, testing intelligently across an organization's interconnected systems and devices, data sources and underlying infrastructures is absolutely critical.

Take a look at the scenario below. This end-to-end process represents a user making a flight reservation.



In this flight reservation scenario, the following steps occur:

1. Customer makes a flight reservation on the airline website (using any standard browser)
2. Reservation is created in the global distribution provider (GDPs) system
3. Customer calls airline to make a flight change
4. Airline agent opens an application called "Sabre Interact" to verify the details of the reservation (this app is running through Citrix)
5. Airline agent makes a change on the application running via Citrix
6. Customer logs in to the website and verifies the change via email

To automate this end-to-end business process, we would use a combination of Tosca's native engines and Vision AI including capabilities like Service Virtualization and Test Data Management:

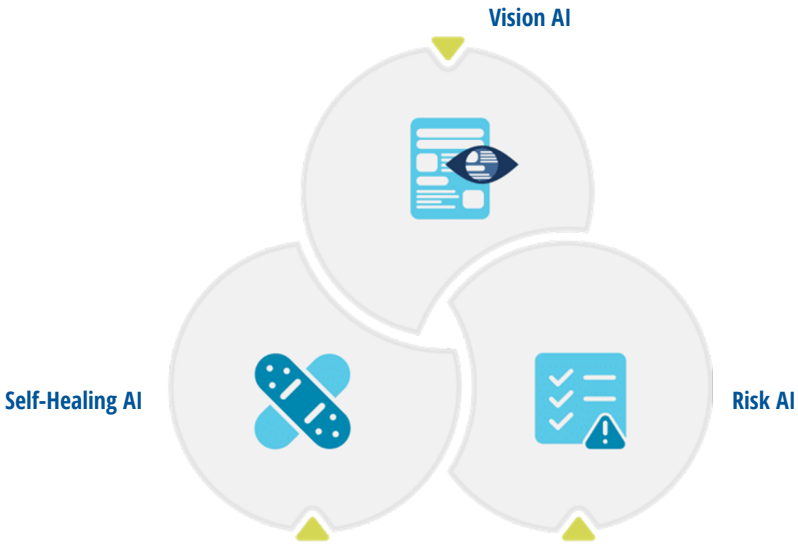
Service virtualization – the global distribution provider (GDPs) is a third-party application that can be unavailable or difficult to access during testing. By simulating this system, i.e. mocking the actual service through a series of API calls and responses, you can to test much faster and provide continuous quality feedback for integration testing.

Test data management – you need to make sure that the dynamic data (i.e. flight reservation number) and static data (i.e. customer ID or address) is consistently masked and updated in all systems throughout our end-to-end testing. This will prevent data conflicts or inconsistencies, and ensure more efficient testing in a distributed agile environment.

Vision AI – you need to create robust automated testing of the Sabre Interact application that is hosted on Citrix. Vision AI utilizes the power of machine learning to recognize any object on a screen, regardless if it is being streamed remotely through Citrix or via other means.

TRICENTIS PLATFORM: VISION AI, RISK AI AND SELF-HEALING AI

Tricentis embeds AI capabilities across the portfolio in order to deliver a comprehensive strategy that will allow agile and DevOps teams to keep up with application changes, and take the pain out of test design, execution and maintenance.



Vision AI – Automatically recognize and identify user interfaces elements and controls across any form factor the same way humans do to aid in the automated generation of robust test cases. This technology is being made available for production deployment with Tosca 14.0.

Risk AI – Automatically detect most at risk objects and select the right set of tests to minimize business and technical impact of code changes. This technology is available for SAP and delivered by Tricentis LiveCompare today.

Self-Healing AI – Test cases automatically adapt as applications evolve with each iteration. The first iteration of this technology is being introduced with Tosca 14.0 as well.

GLOSSARY

The glossary below defines key concepts and terms used exclusively in context with this whitepaper and Tricentis Tosca.

End-to-end testing

Refers to the process of testing how an application interacts with different interconnected and dependent systems, components and applications across an enterprise core business processes.

Identifier / object identification

An identifier identifies the object that we want to interact with during automation by getting its required technical information. This allows Tosca to steer those objects during automation. For instance: click a button, verify data in a table cell, or type text into an entry field. In Tosca XScan, you can identify objects in your test application by using multiple identification methods. The identification of objects by their properties is the default identification method in Tosca XScan. However, if your objects cannot be identified uniquely, Tosca offers are other ways to identify them, such as identification via anchor, index or an image.

Model-based test automation

Tosca uses a model-based test automation approach which involves scanning the application under test, extracting its technical layer and creating a business readable automation model through a no-code approach. The automation models are called modules in Tosca, whereas the method for creating codeless automation is referred to as model-based test automation.

Module

Modules are the building blocks of your tests. They contain the technical information that Tosca needs to navigate and interact with your system under test. For instance: all available options in a drop-down menu, the headers of an Excel table, or how to select a check box.

Object / control / element

Your application under test has various elements that Tosca needs to interact with or steer during automation. For instance: buttons, text fields, tool bars, cells in tables, SAP dialog windows, etc. These are called objects / controls / elements, which we use interchangeably in this whitepaper.

Standard module

Say you want to log into an application, open a URL or close a URL. These are common execution tasks required for automation testing. Instead of creating these modules from scratch, with Tosca, we have standard modules already available for your instant use. These come with the standard subset. We have standard modules for SAP, test data management, TBox automation tools and many others.

Vision AI

Is an AI-powered engine and a patented machine learning technology within Tosca version 14 and above that uses such as convolutional neural networks combined with advanced heuristics to deliver stable, self-healing, platform agnostic UI automation.

ABOUT TRICENTIS

Tricentis is the global leader in enterprise continuous testing, widely credited for reinventing software testing and delivery for DevOps and agile environments. The Tricentis AI-based, continuous testing platform provides automated testing and real-time business risk insight across your DevOps pipeline. This enables enterprises to accelerate their digital transformation by dramatically increasing software release speed, reducing costs, and improving software quality. Tricentis has been widely recognized as the leader by all major industry analysts, including being named the leader in Gartner's Magic Quadrant five years in a row. Tricentis has more than 1,800 customers, including the largest brands in the world, such as Accenture, Coca-Cola, Nationwide Insurance, Allianz, Telstra, Dolby, RBS, and Zappos.

To learn more, visit www.tricentis.com or follow us on LinkedIn, Twitter, and Facebook.

AMERICAS

2570 W El Camino Real,
Suite 540
Mountain View, CA 94040
Unites States of America
office@tricentis.com
+1-650-383-8329

EMEA

Leonard-Bernstein-Straße 10
1220 Vienna
Austria
office@tricentis.com
+43 1 263 24 09 – 0

APAC

2-12 Foveaux Street
Surry Hills NSW 2010,
Australia
frontdesk.apac@tricentis.com
+61 2 8458 0766

v. 0621