APPLICATION NOTE

Testing the Product Lifecycle Management (PLM) Process with Keysight Eggplant Test

Automated testing of PLM processes and product design software

KEYSIGHT

Table of Contents

Navigating the Landscape of PLM Software Testing

Product Lifecycle Management (PLM) systems play a crucial role in enabling high-tech and electronics companies to develop and deliver highly innovative, timely, and cost-effective products that succeed in a distributed global marketplace.

PLM benefits every department involved in a product's lifecycle by providing the correct information at the right time to help define strategies for production, planning, control, and revenue. But these benefits are only realized if your PLM system is installed, configured, and integrated correctly. Worse, errors in its deployment and configuration can have a significant negative impact on the whole business.

There are three important aspects of using a PLM system that incur significant risk if not correctly addressed:

Ensuring accuracy throughout the PLM landscape

This is to ensure that the PLM system and any configurations (customizations/business logic) work as expected. The logic can range from simple rules to complex custom objects and all need to execute accurately and consistently. Once the PLM system is configured and tested, its behavior, logic, and data will be integrated into other parts of the ecosystem. Ensuring these integrations are consistent and that the values and triggered business logic contained within these 3rd party systems behave correctly is as critical as the core PLM system itself.

A fundamental premise of deploying a PLM system is to ensure consistency. A data point should only be entered once and then referenced across all systems. This is designed to ensure that data and calculations are uniform within the PLM and across all digital systems (MES, CAD, ERP, etc.), and all artifacts including technical documents (PDF), emails, etc.

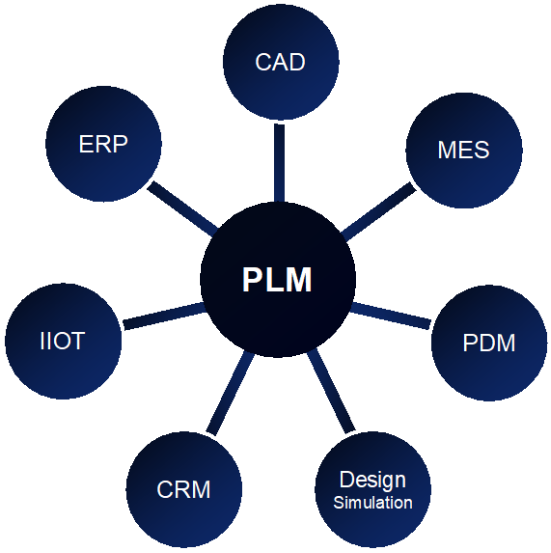

Figure 1. Product Lifecycle Management (PLM) applications and commonly supporting ecosystems.

Guaranteeing consistency during changes and upgrades

Changes to the logic within the PLM system and/or any of the supporting systems can negatively impact the business, this can include erroneous data points that can quickly expand into supply chain and compliance issues.

All systems are updated and upgraded increasingly frequently, new features and security updates force the installation of newer versions of the PLM system and the wider ecosystem. Even minor service packs can inadvertently impact previously working behavior.

The PLM system's and associated ecosystem's behavior must be confirmed and validated (normally through regression testing) following any changes before they are placed into production.

Maintaining performance and availability for optimal usability

An organization will place a huge demand on a PLM deployment, commonly large numbers of users depend on its capability, but also its performance and availability. An unavailable or unacceptably slow PLM system (such as when loading large files) harms productivity and incurs frustration across the user base.

It is important to know that a deployed system will support the load and needs of expected users before putting into production; then, once confirmed continued monitoring is important to ensure performance regressions and availability dropouts do not impact the business.

Addressing These Problems Today

PLM platforms are complex ecosystems consisting of multiple applications operating in heterogeneous environments and all are offered in a mix of "on-premises" and/or as hosted "SaaS" platforms. To guarantee the level of software quality expected by the users of your PLM environment, it is necessary to define and execute a set of tests that cover both initial deployment and ongoing regression scenarios. Today, these are almost exclusively carried out manually.

The cost, complexity, and speed of manual testing put significant pressure on making any modifications or improvements. This results in limited test coverage (exposing dangerous gaps) and adds significant friction to the business's need to implement continuous changes and updates. The only viable solution to speeding up production releases is to embrace automation.

The question of how to move from a manual testing strategy to one of automation must consider both the specific needs of CAD, PLM, and MES applications and the various supporting systems that often include ERP, CRM, or a simulation application. And, most importantly, the needs of the business to drive value from these systems.

Eggplant Test for Automated PLM Testing

Testing PLM processes in design and manufacturing industries is challenging due to the graphical nature of design applications and the complexity and breadth of the other systems involved.

Keysight Eggplant Test (hereafter called “Eggplant Test”) can drive end-to-end tests through all these systems. Eggplant Test’s innovative solution uses computer vision to emulate a real user’s view of the systems providing both:

- Automation and visual verification without manual intervention, regardless of the visual complexity (e.g. moving sliders, interacting with multiple pop-ups, counting the number of holes visible on a bracket and interacting with each individually, etc.). This can operate (automate and validate) across both 2D and 3D user interfaces.
- The ability to automate and test any platform or the technology it is written with (web, mobile, native app, custom app, PDF documents, IIoT, ARKit, etc.).

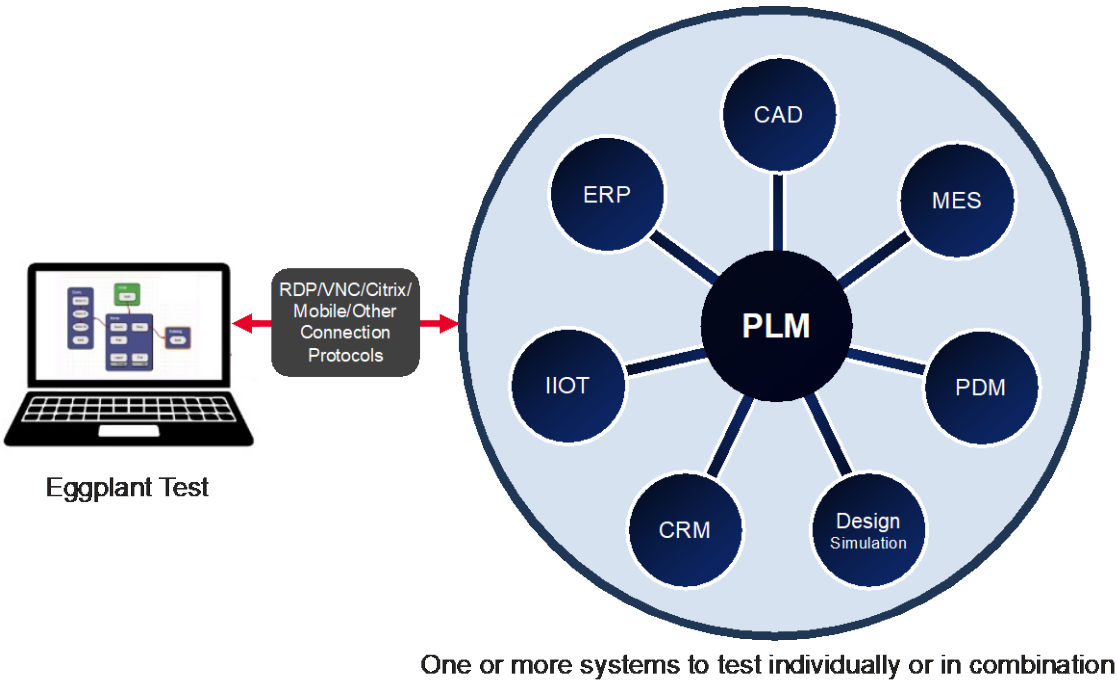

Figure 2. Eggplant Test, driving end-to-end tests of PLM processes starting and ending with any system.

Eggplant Test focuses on automating and validating the user interface without needing any connection to the object (or API) layer and therefore does not require any prior study of the code structure of the applications tested; the tests are defined in a manner that directly reflects the way users would use them.

Using Eggplant Test to Ensure Accuracy Throughout the PLM Landscape

Computer-aided design (CAD) software creates a “digital twin” of a physical product enabling designers to create, modify, and manipulate a product digitally. CAD software interfaces include sliders, dials, and two- and three-dimensional canvas controls for manipulating designs.

Eggplant Test makes automated testing of this design manipulation easy by providing a language with intuitive commands that can interact with the controls the way a user would. This approach removes the need for complex navigation of the structure of underlying UI technology (DOM), dramatically simplifying the process.

When design changes occur, Eggplant Test can also verify the changes. The image and text recognition capabilities of Eggplant Test are perfectly suited to confirm that the expected changes were made correctly. Eggplant Test intelligently understands the screen using computer vision, and by mimicking a user’s view of the system, Eggplant Test provides invaluable visual verification testing without manual intervention.

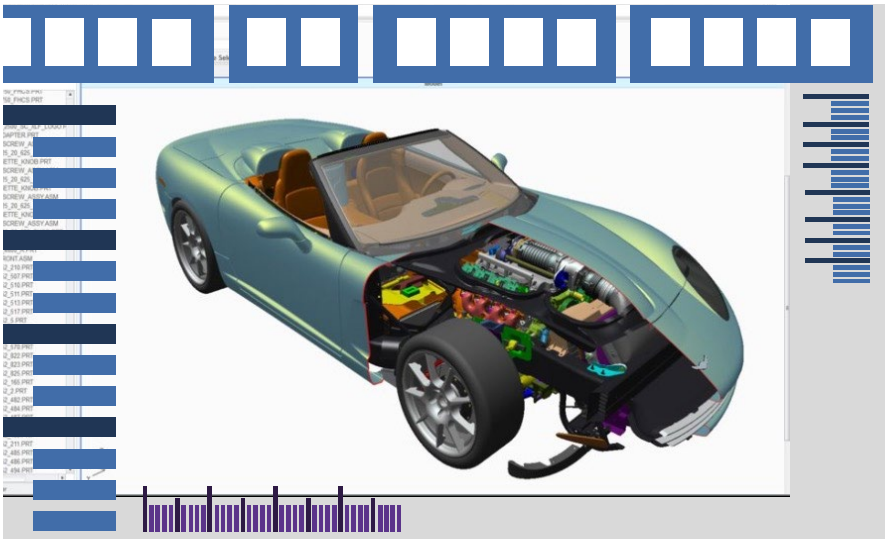

Figure 3. A representation of an automobile design in a CAD application with complex dynamic control interfaces and a mix of 2D and 3D visualizations – all automatable and verifiable with Eggplant Test.

Connecting to PLM and the ecosystem

In Eggplant Test, the connection to any system uses the same simple premise of the "**connect**" command. Once connected, any future commands will execute on that target system. Eggplant Test uses a low-code natural language-like system with commands that are easy to understand with or without programming skills. For example, to connect to a desktop that is running the PLM system, we first connect to it, and then to ensure we're ready to start a test we can check to see if the Start Menu has loaded, to do this we can use this simple example:

```
// Connect to a remote desktop and wait until we see the start menu icon before continuing

connect "Remote_PLM_System"

waitfor 4, "start_menu"
```

Figure 4. Eggplant Test script for connecting to a system under test and then waiting for the start menu to appear. In this example if the start menu takes more than 4 seconds to appear we will fail the test. This allows us to ensure we don't wait indefinitely for something to happen.

When Eggplant Test establishes a connection, you see the viewer window open displaying the desktop it is connected to (which can be remote or the same one it is running on).

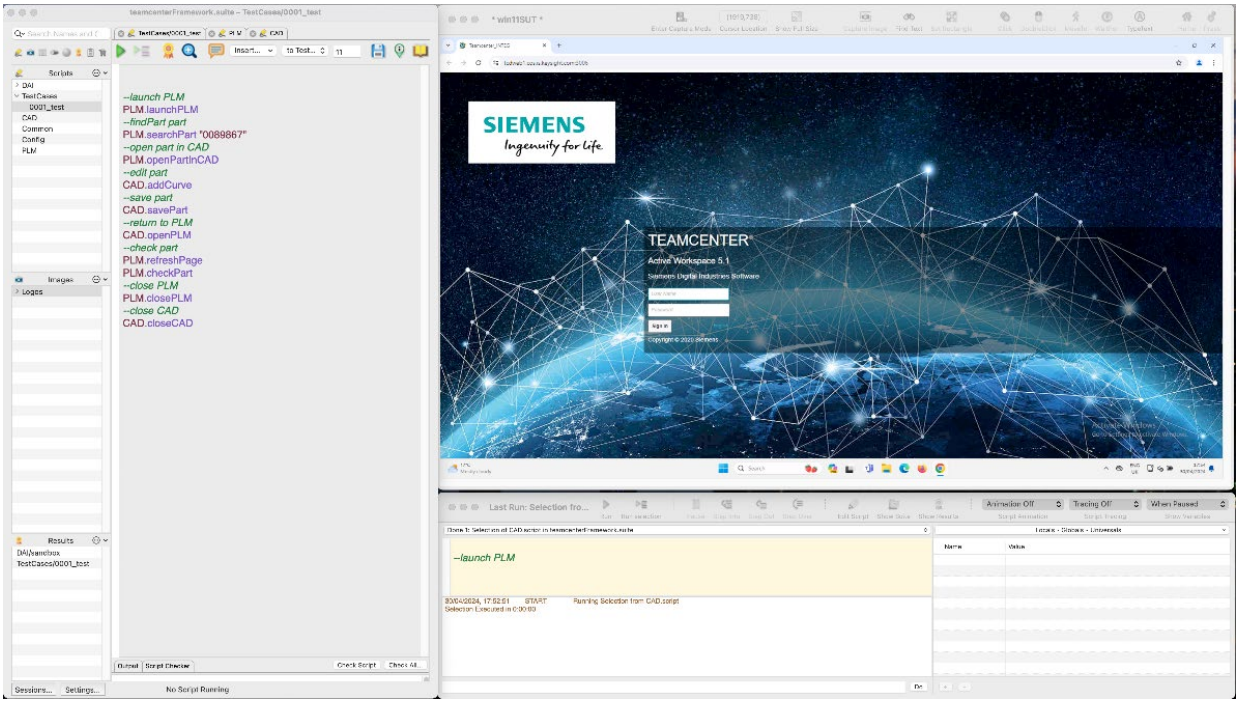

Figure 5. Eggplant Test connected to a desktop ready to start a test.

Testing the ecosystem with true end-to-end testing

The use of the "**connect**" command in this way is very powerful as we can easily swap between different systems in a single test. This powerful technique can easily compare or transfer data items across any mix of systems. For example, we can connect to the PLM system, and read the value of a temperature threshold (defined in a region that we specified (not shown below)). Then we can connect to the CAD system, find the equivalent reading, and simply compare the two numbers. In this case, Eggplant Test can understand and "read" the visual screen to generate two numeric values for comparison:

```
--Connect to PLM System
connect "PLM_System"
--Read and store the temperature value
put ReadText(temperature_Box_PLM) into tempReadingPLM
--Connect to CAD System
connect "CAD_System"
--Read and store the temperature value
put ReadText(temperature_Box_CAD) into tempReadingCAD
--Compare the values in each system
assert tempReadingPLM is tempReadingCAD
```

Figure 6. Connecting to two separate systems as part of a single test. This powerful technique enables simple, true end-to-end, testing across all required systems.

Manipulating complex PLM/CAD software systems

Using Eggplant Test scripts, you can send commands to your test system to simulate user manipulation of CAD controls. Creating scripts in Eggplant Test is easy to understand and simple to use. Consider the following sample list of commands that you can use to manipulate user controls:

Commands

- Click, double click
- Scroll up, scroll down
- Drag & drop
- Type text
- Read text
- Swipe

For example, you can use “click” and “drag” commands to manipulate a product in a CAD digital canvas (or any of the controls in CAD software) to see the product design in different views. This includes interacting with complex layers of popovers, pop-ups, and sidebars. Note that the Eggplant Test scripting language also includes commands specifically for mobile device gestures, such as “swipe.”

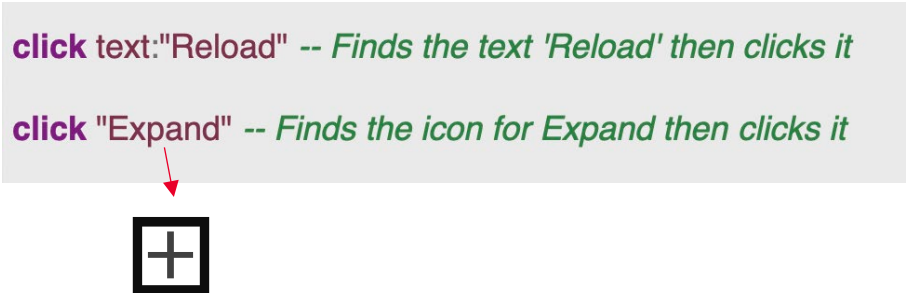

Figure 7. Two simple Test commands. Topmost clicks the “Reload” button by looking for the text “Reload”, and the lower clicks the “Expand” icon (with the representation of the icon shown) by searching for the icon on-screen.

We can interact with these graphically rich systems in a wide variety of ways, not just view properties but to interact with the design itself; we can select items, modify the properties, add new items to the design (such as a new slot or element), and modify those new elements (such as changing the radius of a curve). Because Eggplant Test can mimic the interactions of a real user, it can automate the manipulation of the design interface in the same way.

Interacting, automating, and verifying designs

When an Eggplant Test script sends commands to manipulate the controls in a CAD application, it can also send commands to verify the changes made by those commands. Eggplant Test can “look for” an element on the test system to ensure the expected change occurred.

For example, Eggplant Test can verify that any changes in a design are correctly rendered after they are applied (in the CAD system and then any other system). For example, we can:

1. Visually detect, and find the location of, an element in the design (such as a slot).
2. Delete the slot object.
3. Verify that the slot has been deleted.

Eggplant Test can report the success or failure of the delete action.

The following sample script performs these tasks:

```
-- Find the slot an screen and click on it to cause the context menu to pop up
Click "slot"
-- Find and click on the 'Delete' item pop-up menu
Click Text:"Delete"
-- Finds and click an the "OK" pop-up
Click Text:"OK"
-- Check if the slot is still visible
Put ImageFound(image:"slot") into confirm_deleted
-- Report an error if the slot is still visible
Assert confirm_deleted is false
```

Figure 8 An Eggplant Test script that finds an element, deletes it, and then checks that the element was deleted. A visual representation of the slot is shown. We can expand this script to report on whether the socket was found in the first place.

We can extend this use case to delete all slots, this is simply done using the "**EveryImageLocation**" command. This, when given something to search for (some text, a number, an image, part of an image) will return the locations of each of these found. With that list, we can simply count the number of entries (useful to confirm the number of slots (or other graphical elements) that should visually appear on a design) or we can iterate through that list and delete the slots (as above) one by one in an automated way:

```
repeat with each item currentSlot of EveryImageLocation("slot")
    click currentSlot
    click text:"Delete"
    click text:"OK"
end repeat
```

Figure 9 Find the locations of all visible slots and delete them in the CAD system, using the same commands and method that a real user would. "EveryImageLocation" is useful for accurately counting the number of visual elements that should be present in the design.

Validating the assets

Most of the systems used alongside PLM are digital business systems such as CAD, ERP, etc., but there are ancillary elements that are just as critical – these include PDF documents, Word files, spreadsheets, plain text files, API calls, and data in databases. It is just as important that the data contained within these also match what is within the digital systems.

In the same way that Eggplant Test can connect to two different digital systems and compare values; it can just as easily analyze and validate the content of these documents too, such as initiating the creation of a PDF from any system and validating each data element within that document whether the data is numeric, textual or graphical. This can be done against the digital system and/or a prior baseline PDF document.

```
CompareScreen "design_Reference", rectangle:"designPage"
```

Figure 10 Example command to compare two screens to highlight the differences.

Using Eggplant Test to Guarantee Consistency During Changes and Upgrades

So far, we have covered how Eggplant Test makes it simple to run basic or complex automated tests within one, or across many, systems. The intent of these use cases is focused on correctness (or functional testing). Ensuring that the system works as expected in all cases is critical to a PLM deployment, and given the nature of PLM systems, this makes it critical to the business.

It is very difficult to ensure that nothing changes within the PLM ecosystem as the changes can be out of the team's control, this can include relying on cloud/SaaS systems that get updated regularly by the vendor, operating system updates automatically applied, or security patches being forced into systems. So, even with great control, the environment will change.

Add functional upgrades to systems, such as upgrading a system from V10 to V10.1, and the changes, and associated risk, to the current deployment are more significant.

There is an obvious and beneficial solution. This is to re-use the assets used to correctly test the initial deployment. There are two aspects to consider:

1. Deciding which automated tests to run.
2. Updating any of the assets to adapt to the new install or update.

Deciding which automated tests to run

Without automated testing any test cycle is costly and slow – this puts pressure on not updating systems, which incurs its risks as some systems update anyway (cloud, operating systems, etc.); not applying security patches exposes more fundamental risks; and not updating new functionality can give the advantage to your competition.

In the simplest case, we can define a core set of tests that cover the critical use cases across the PLM system. These act as "smoke tests" to ensure the basic, fundamental functionality continues to work. In, addition, sets of tests, known as "regression packs," can be added as needed. These normally focus on specific areas for simpler management.

Such tests are powerful in Eggplant Test as they can easily cover many systems in one test case, removing both the need for manual testing of any one system *and* the need for manual testing to cover the gap or interactivity between systems. Eggplant Test can truly execute tests across a range of systems just as a real user would, regardless of technology or visual complexity.

For example, we can verify PLM data across ERP and CAD systems simply. In this example, we validate that a data item is found on both the ERP system and CAD system; the flow is:

1. Connect to the main PLM system and launch the ERP application, then wait for its logo to appear before continuing.
2. Search for the product ID in the ERP system and alert if not found.
3. Connect to the main Design system and launch the CAD application, then wait for its logo to appear before continuing.
4. Search for the product ID in the CAD system and alert if not found.
5. Disconnect from all systems.

```
// Connect to PLM system
connect "PLM_system" --PLM system connection name
waitfor 4, "start_menu" --Windows Start menu verifies successful connection

//Verify Data in ERP desktop application
doubleclick "erp_icon", waitfor: "erp_logo" --launch erp application
click "search_field"
validatePID --call handler to validate the product id is found

// Connect to design system
Connect "Design_system" --design system connection name
waitfor 3, "start_menu" --Windows Start menu verifies successful connection

//Verify data in CAD application
doubleclick "cad_icon" --launch CAD application
waitfor 5, "cad_logo"
click "search_field"
validatePID

//disconnect from all SUTs
repeat until connectioninfo().status is "not connected"
    disconnect
end repeat

to handle validatePID
    typetext pid && returnKey --enter product id and press return to search
    If ImageFound(text: pid)
        LogSuccess "Expected product id found"
    else
        LogError "Expected product id not found"
    end If
    click "close_btn" --Close application
end validatePID
```

Figure 11. Note that this example uses a “handler” called “**validatePID**”, this is a reusable function that can be used elsewhere in the script. In this case, the “**validatePID**” handler will search for the name of the product ID in a search box and then look for a resulting match in the results. This generic function can be used across all different systems.

A digital twin of your digital twin

To take end-to-end testing of PLM systems further, Eggplant Test supports a graphical digital twin approach to test design. The graphical modeling environment allows for test conditions to be defined graphically using a state chart-like notation. The result is a workflow that allows for quick and easy definition of not just explicit paths, but also all viable paths – allowing for all possible conditions and use cases to be captured quickly.

Three elements represent the model:

- State: The blue boxes that define a specific context (often used to represent a screen or mode), an example state may be “Editing the properties of a CAD model element”.
- Actions: The grey boxes within the states that define activities that can happen within that state, for example, “delete element” or “edit element”.
- Snippets: These are the automation assets, in essence, small scripts as in the examples above, that can implement the intent of the actions.

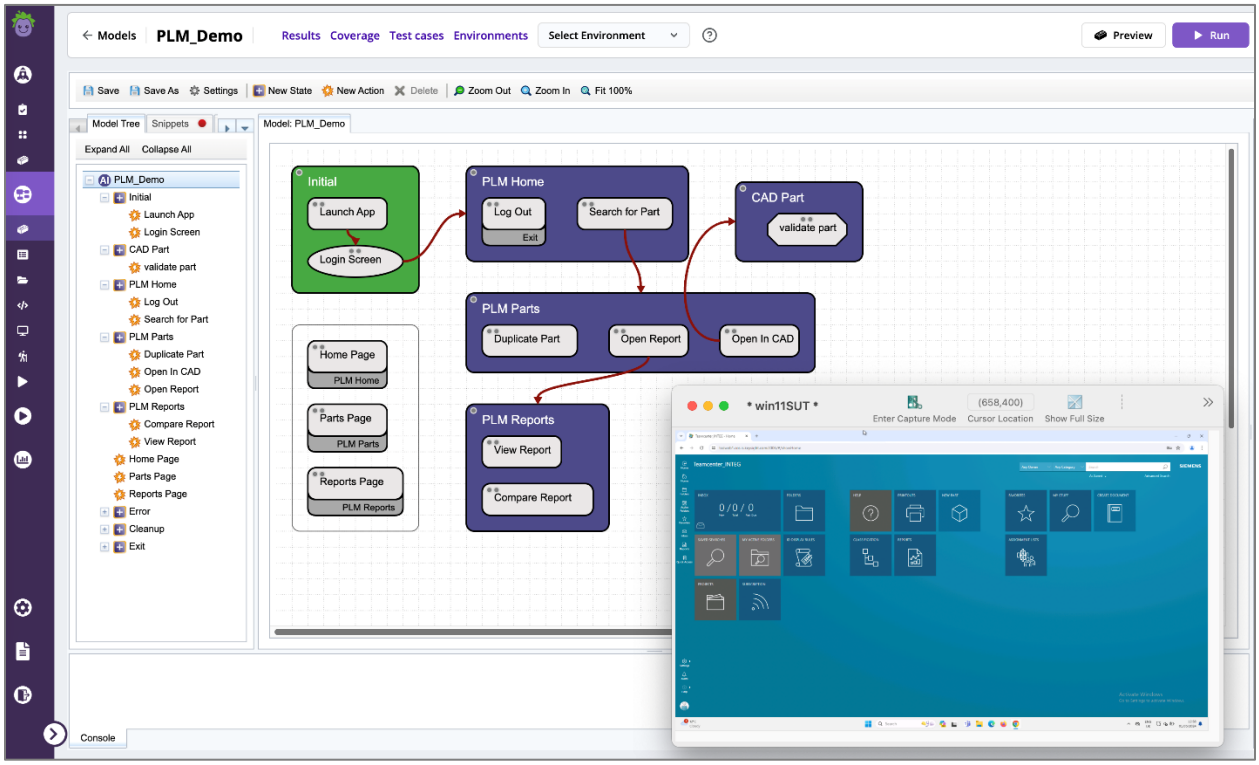

Figure 12. Eggplant Test running snippets to test workflows in the model of a PLM system, with the PLM system itself shown in the foreground (lower right).

A model can represent one system and/or any combination of systems and can be used in two modes, which can be used interchangeably:

- A canvas to define specific (fixed) test cases – this is simply a matter of clicking the sequence of actions needed to define a specific test – this is the equivalent of a test written with a script, though more flexible and simpler to visualize.
- AI-augmented testing with the Eggplant Test AI test generation engine. This technology utilizes an ensemble of AI algorithms to efficiently auto-generate the set of most valuable test cases to execute based on what it has learned. The algorithms include those focusing on maximizing coverage, those focusing on recent changes, and those actively hunting for defects and bugs in the systems.

Regardless of the test design philosophy: test case scripts or models, or a combination of both – the execution utilizes the same technology we have covered previously. Eggplant Test's ability to understand and interact with any system designed to be used by humans provides a simple, accurate, fast, and robust means to test, validate, and automate PLM systems.

Updating any of the assets to adapt to the new install or update

While any update can change the behavior and or structure of a system, these are commonly only significant during major upgrades such as migrating from V11 to V12. Eggplant Test's approach to testing (and automation) provides specific capabilities that take the pain away from updating any test assets:

1. Commonly, the UI does not change as much as the internal structure and because Eggplant focuses on the UI it is far less impacted. A vendor could rewrite their UI system from ANGULAR to REACT, and while this would require a full re-write of the tests in other systems, it is irrelevant to Eggplant Test – we only need to focus on any UI changes. As vendors strive to maintain consistency within their user experience the resultant changes are often small.
2. Powerful Computer Vision technology within Eggplant Test to find and control elements on the screen is not limited to specific pixel-by-pixel arrangements. For example, we can capture an icon on the desktop application and use this same image to find the same icon on a mobile app. The image may look the same to a human but can be of radically different sizes and use different rendering engines. In another example (below), a CAD application has been updated and the "Export" button at the bottom of the page, has moved to the ribbon and changed both its font and color. If we ask Eggplant Test to click on the "Export" button it will do that regardless of where it is (it is also possible to limit it to a certain region if needed) and so continues to work seamlessly. For cases when UI elements have dramatically changed, we provide a concept of "image sets" to cover different representations and intelligent technology for automatic, or semi-automatic, "healing" of test assets.

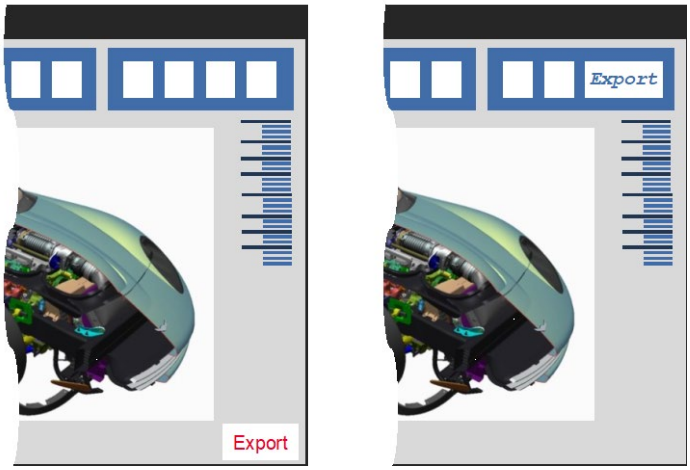

Figure 13. Two different versions of a CAD system where the “Export” button has changed location, font, and color – none of which impact Eggplant Test or the need to change any test assets because our computer vision approach does not need access to any web/DOM or API data.

Using Eggplant Test to Maintain Performance and Availability for Optimal Usability

Functional testing of the PLM landscape forms the initial focus of ensuring systems are deployed correctly and work together as required. However, once deployed, the availability and performance of these systems are just as important.

Simple monitoring will tell you if your hosts are running, and even if the applications are responding – but none of these mirrors the needs or requirements of the users who rely on these systems. The non-functional demands of the users include:

- Time taken to load specific models (high and low complexity).
- Time taken to pass data across different systems.
- Time taken for a full-page refresh of a complex page.

In all these cases (and others), accurate data can only be gained by understanding the visual status of the screens. Eggplant Test provides benefits in two ways:

1. Ability to define the expected result from the user’s perspective, e.g. screen loaded is when the model is fully visible and the first “move” command results in an actual change to the model.

2. Ability to record accurate and detailed timing data.

```
put the time into startTime
        -- any activity or script here
put the time into stopTime
log "That took " && stopTime - startTime && " seconds to complete."
```

Figure 14. Example of a simple timing script. The test to time can sit in the location of the green comment. Any visual verification capabilities can be added here to ensure the timing data records results that are meaningful to real users.

Eggplant Test doesn't only log but can compare against baselines or thresholds and take specific actions including sending alerts, notifications, or data to operational dashboards. This allows response time metrics to be made available to create reports to monitor the performance of your CAD, PLM, and MES applications.

Reducing PLM Friction and Risk with Eggplant Test

This document has covered the fundamentals and risks of deploying and managing a PLM system within any organization. In summary, these fall into three categories:

1. **Ensuring Accuracy Throughout the PLM Landscape**

   Making sure the systems work correctly in isolation and when connected.

2. **Guaranteeing Consistency During Changes and Upgrades**

   Making sure the systems continue to work correctly as systems change or get updated.

3. **Maintaining Performance and Availability for Optimal Usability**

   Making sure the systems work for your users all the time to maintain productivity.

We have also described how Eggplant Test's unique approach is squarely aimed at addressing these challenges. This high-level document is intended to provide a practical illustration of Eggplant Test's concepts. There are additional capabilities that are out of scope for this document but fall into the breadth of Eggplant Test's capabilities. These enable:

- Scalability of testing, whether for concurrent functional tests, performance load tests, or monitoring purposes.
- Connect to any system (any PLM platform, CAD, ERP, …), regardless of technology (web, desktop, IoT device, …) or deployment system (Citrix, RDP, Local, …), current and future.
- Migration of systems (such as from one PLM platform vendor to another).
- Integration into 3rd party systems, such as CI/CD pipelines, databases, etc.
- Scalable services and customer success teams to provide any level of support and guidance.

Conclusion

Eggplant Test provides our customers with a future-proofed platform that continues to work while the systems under test evolve. This normally involves changes to the underpinning of an application (such as an application major version upgrade), but also the whole experience (such as moving from a 2D desktop application to a 3D Virtual Reality headset); Eggplant Test provides assurance, regardless of what the future holds.

In the words of our customers:

- “What used to take 1 month for manual releases now takes 4 hours.”
- “We turned 15 days of manual testing into an overnight test.”
- “Eggplant Test enabled the team to move from completing 10-hour test event every three days to running a 24-hour test event every day.”
- “Experienced testers that didn’t have much automation experience appreciated how the solution could complement their abilities and improve the way they work.”