What Is Graphic User Interface Testing?

Inflectra: Software Built For You

Our one goal is to help you succeed. We care deeply about giving you the best quality service and support you've ever had.

As many users, projects, tests, releases, items, API calls as you want. All pricing is based on concurrent users.

Flexible options to make your life easier. Use on desktop or mobile; your servers or our cloud, sensible add-ons, fairly priced.

Why Choose Rapise for GUI Testing?

Rapise® provides powerful and easy to use automated testing. When you need to test web, mobile or desktop applications or test your APIs and web services, Rapise makes it easy. Rapise is ideal for today's agile software projects.

- **Web, Mobile & Desktop Testing in One Tool** – Rapise comes out of the box with support for testing Desktop, Mobile and Web Applications. It allows the same test scripts to be executed in multiple browsers unchanged and supports most GUI frameworks
- **Learn and Go Testing™** – Rapise's Learn and Go testing is much more efficient than traditional record & playback. Objects are edited during the learning process vs. at the end. Rapise significantly reduces testing time and **gets your application to market faster**
- **Script or Scriptless Testing – Your Choice** – Rapise includes both an easy to use scriptless test creation language that can be used by domain experts and a full JavaScript IDE for programmers who want the full power of a test script language.

4 Top Things to Look for When Choosing a GUI Tool

1. An authoring framework that allows you to write tests, make changes, find issues and be able to deploy the tests on all the environments you need to test. Depending on the preference and skill level of your testers, choose scriptless or script-based tools.
2. A tool that is well supported by the manufacturer and is keeping up to date with new web browsers, operating systems and technologies that you will need to test in the future.
3. An object abstraction layer that allows test analysts to write the tests in the way most natural for them and that lets automation engineers to create objects that point to physical items in the application that will be robust when re-sorting a grid or adding data to the system.
4. Support for data-driven testing since one of the big benefits of automation is the ability to run the same test thousands of times with different sets of data.

Visit us at www.inflectra.com for a free trial
Or get in touch with us: sales@inflectra.com, 1-866-572-5878 or +1 202-558-6885 (international)

inflectra®

What Is Graphic User Interface Testing?

Graphic User Interface (GUI) Testing is the process of ensuring proper functionality of the graphical user interface for a specific application. This involves making sure it behaves in accordance with its requirements and works as expected across the range of supported platforms and devices.

Why Is GUI Testing Important?

Modern computer systems are generally designed using the 'layered architecture approach', which means that the core functionality of the system is contained within the "business logic" layer as a series of discrete but connected business components. They are responsible for taking information from the various user interfaces, performing calculations and transactions on the database layer, and then presenting the results back to the user interface.

Testing of the GUI allows testing functionality from a user's perspective. Other types of testing would miss a user interface failure, so GUI testing is an important part of the testing toolset.

What Types of GUI Testing Exist?

There are two main types of GUI testing available: **Analog Recording**; and **Object-Based Recording**.

Which Should I Use?

The best practices are to use Object-Based recording that is more robust and reliable where possible and then use Analog recording to "fill in the gaps" where nothing else works.

Analog Recording

This is often what people associate with GUI testing. Analog recording captures specific mouse clicks, keyboard presses and other user actions and then simply stores them in a file for playback. For example, it might record that a user left-clicked at position X = 500 pixels, Y = 400 pixels or typed the word "Search" in a box.

The benefits of analog recording:

- It works with virtually all applications regardless of the technology or platform used
- It is quick to write tests
- As long as the GUI is stable and you can guarantee the screen resolution and window position, it can be a good way to test older applications.

The major drawbacks to analog testing:

- Tests are sensitive to changes in the screen resolution, object position, the size of windows, and scrollbars.
- Tests are not intelligent; they don't look for specific objects so you change the UI in any way (e.g. change a button to a hyperlink), the tests may need be rewritten. Analog tests are therefore considered "brittle"
- Human validation is essential. When analog tests perform an action it receives very limited feedback: the testing tool can only click at a coordinate or send a keypress and can not understand if the application worked correctly.

Object-Based Recording

With object-based recording, the testing tool connects programmatically to the application being tested. It "sees" each of the user interface components (a button, a text box, etc.) as separate entities and can perform operations (click, enter text) and read the state (is it enabled, what is the current value) reliably, regardless of where that object is on the screen.

The benefits of object-based learning:

- Test are more robust and do not rely on UI objects being in a certain position on the screen or being in the top-most window, etc.
- The test will give immediate feedback when it fails
- Tests are less "brittle" and require less rework as the application changes. This is especially important for a system under active development and UX changes.

The drawbacks to object-based learning:

- The testing tool needs to have specific support for each of the technologies being used. For example, if you have a web page that contains a Java applet, you need a testing tool that understands this technology
- For some technologies (e.g. Flex, Flash) "instrumentation" must be added to the code so that testing tools can "see" UI objects. If you don't have access to the applications's source code, it's not possible to test it
- Writing tests takes more skill than simply clicking and pointing: a tester may need to use Spy and inspection tools to navigate the object hierarchy and to interact with the right UI elements.

The Challenges with GUI Testing

Regardless of the approach taken, there are some general challenges with GUI testing:

- **Repeatability** – Unlike code libraries and APIs which are normally designed to be fixed, user interfaces tend to change significantly over time. So, when testing a user interface large parts of the test script may need to refactored for it to work correctly with the updated version
- **Technology Support** – Applications may be written using a variety of technologies (e.g. Java or .NET) and control libraries (Java AWT vs. Java SWT). Not all libraries are as easy to test and specific testing tools may work better for some libraries than others
- **Stability of Objects** – When developers write an application, their choices can determine how easy an application's GUI is to test. For instance, it is much harder to test an application where objects have a different ID value every time a window is opened
- **Instrumentation** – In some cases, testing requires developers adding special code (called instrumentation) or including a specific library. If you don't have access to the application source code in such cases, testing options are limited.

What Are the Best Practices for GUI Testing?

1. Upfront Technology Assessment

Before selecting a tool, it is generally recommended to perform an upfront assessment of the technology, UI controls, and widgets the application uses. Often there are some non-standard UI controls and elements that testing tools may struggle to interact with.

Different technologies in the application may require different testing libraries. For example:

- Microsoft Active Accessibility (MSAA) - This Microsoft API was originally developed to enable screen-readers and other assistive technologies. It was adopted by testing tools as a reliable way to see GUI objects
- Microsoft UIAutomation - This is the Microsoft's replacement API to MSAA. It allows testing tools to connect to and perform operations on objects in the Windows GUI
- Java Reflection - for applications running in a Java VM that use either AWT or Swing GUI libraries, testing tools need to connect to the running Java VM and use the reflection API to understand and test the GUI
- .NET Reflection - for applications running in the .NET CLR using 100% managed code, testing tools can connect to the CLR and use the reflection API to get more precise information than via MSAA or UIAutomation
- Selenium WebDriver - The WebDriver API is available on most browsers to give programmatic access to send commands to the browser and inspect its Document Object Model (DOM). It does not, however, let you record user actions
- Web Browser Plugins - Most (but not all) web browsers support a plug-in architecture that lets testing tools interact with the loaded web page, inspect the browser DOM, but unlike WebDriver, also record user interactions
- Apple XCUITest – This is the current testing API provided by Apple for iOS devices. It was introduced in Xcode 7 and can record user events
- Android UIautomator – This is the current testing API used to test Android applications. It is available on the most recent versions of the Android API and is used by most modern mobile testing tools.

2. Be Realistic About What to Automate

In theory, you could automate 100% of your manual tests and replace them with automated GUI tests. However, that is not always a good idea! Automated tests take longer to write than manual test since they have to be very specific to each object on the page.

If you need to choose which tests to automate, start with the 20% of test cases used most often. These tests will have many hours spent on them: every day, every release, every build. They are also likely those that have the most impact on customer satisfaction. Automating these test cases will reduce overall test time by the greatest factor, freeing the team for other tasks.

3. Avoid the Perfection Trap

Sometimes when automating an application, you will meet a special UI control that is not handled by the testing tool that you are using (or maybe is not handled by any testing tool that you have tried). The solution is to be creative, if you can use analog recording for that one control, then maybe that is the best solution. Or you may need to use a combination of techniques to simulate the user action, using special keystrokes to avoid the button in question.