Originally test tools offered simple capture/playback capabilities: recording and playing keystrokes, then capturing and comparing screens. While simple to create, these tests proved almost impossible to maintain, and capture/playback tools were eventually replaced by test scripting tools.

Scripting tools are essentially specialized programming languages that allow more logic and control in designing tests, including the use of external data sources to provide varying test data conditions.

While more powerful and flexible, scripting tools introduced issues of their own. They required extensive development skills and effort to implement while not necessarily improving maintainability. Unfortunately highly individualized scripting approaches, coupled with a lack of documentation, often resulted in obsolescence or rewrites of script libraries comprising tens of thousands or more lines of code.

Test frameworks were developed to reduce the cost of implementation and ownership by providing pre-written scripts and utilities to support most standard testing tasks and, in some cases, substantially reduce—or even eliminate—the need to learn and use scripting languages.

The primary differences among the commercially available frameworks relates to:

<table>
<thead>
<tr>
<th>Library Design</th>
<th>Whether the underlying execution “engine” or script library is mapped to action words, windows, or classes. These differences, further explained below, will determine the level of development required for implementing an application and the degree of reusability and maintainability of the script library.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case Development</td>
<td>Whether test cases are developed and stored using spreadsheets or databases, and whether they follow a custom or standard format. While spreadsheets may be simple and inexpensive to use, they are difficult to manage and maintain. Relational databases, on the other hand, can cross-link requirements, test cases, test data and application objects to each other, reducing the opportunity for user error and increasing maintainability.</td>
</tr>
<tr>
<td>Application Specific or Portable</td>
<td>Whether the framework functionality is specific to an application or is portable across multiple applications. This dictates the degree of reusability and thus value of the framework investment.</td>
</tr>
<tr>
<td>Implementation and Maintenance Effort</td>
<td>Whether the framework can be licensed and work out of the box, or requires customized development for each implementation. This also affects the maintenance overhead as the application changes. Maintenance is also affected by whether the test case development interface must be changed when the application is modified.</td>
</tr>
<tr>
<td>Tool or Platform Dependent</td>
<td>Whether the framework supports only selected commercial test tools or is tool independent or—in some cases—does not require a tool at all. Also whether the tool supports only a particular platform, such as mainframe or Web.</td>
</tr>
<tr>
<td>Logic</td>
<td>How error handling, recovery and result logging is accomplished. Some frameworks require this functionality to be programmed, while others provide it automatically.</td>
</tr>
</tbody>
</table>
Below is a more detailed examination of each type of framework.

**Action Word Engines**

**Library Design**
Action word engines are mapped to high-level application functions. In this model, test tool experts with development skills either generate or develop scripts that perform application actions such as “enter customer information” or “verify order status”. Test designers, who have application expertise but no programming skills, combine the action words and related test data into tests.

**Test Case Development**
Action word engines may employ either spreadsheets or relational databases for creating test cases but in either case require formats that are custom to the action word. Using spreadsheets make the tests difficult to manage and maintain as it is difficult to enforce naming conventions, and there is no automated means to link application changes to the affected scripts or test cases.

**Application Specific or Portable**
Because the action words capture application tasks, the scripts are unique to the application under test and have no reusability across applications. This requires a substantial up front development effort and ongoing maintenance, which means that the customer must have scripting skills in house.

**Implementation and Maintenance Effort**
Although a library of templates may be available, these must be individually tailored out to each application function. Since there may be hundreds of such functions, the initial development effort is significant. Further, any time application behavior is captured in a script, maintenance is required to the scripts when the application changes, and the test case spreadsheet or database tables may also require maintenance since the format is driven by the data requirements of the action words.

**Tool or Platform Dependent**
Action word engines are likely to be tool or platform dependent because the script drivers and templates are written in a specific scripting language. This may limit the use of the framework to only those platforms that the tool supports, and may also prevent end to end testing of tests that span applications and platforms.

**Logic**
Error handling and recovery must be programmed into the action word scripts as well as to the overall script execution driver. This reduces the test developer's control over options to respond to varying conditions. Action word frameworks are best suited to applications that do not provide user interfaces, such as embedded systems, back ends or APIs which are highly customized and proprietary so that it is not possible to take advantage of standard script functions.

**Window Engines**

**Library Design**
Window engines employ scripts written at the screen, window or page level. For each formatted display, a script is written to provide input and another to verify output for the fields or objects on the screen. Test scenarios are designed using sequences of the windows together with the data values for either input or verification.
## Test Case Development

Window engines may employ either spreadsheets or relational databases, and the format of the test case varies with each window. While using spreadsheets has the same management and maintenance issues as action word engines, they have an advantage in that changes to a particular window are more easily mapped to the affected test case.

## Application Specific or Portable

As with the Action Word frameworks, Window frameworks incorporate application-specific code into the scripts; they are unique to the application under test and have no reusability across applications.

## Implementation and Maintenance Effort

For an application with hundreds of windows or pages, a corresponding number of scripts are required, and each time an application window is added or changed it requires additions or modifications to the script as well as to the spreadsheet or database tables. This means the customer must have scripting skills in house.

## Tool or Platform Dependent

Window engines are also likely to be tool or platform dependent within a test case because the script drivers and window scripts must be written in a specific scripting language. This may limit the use of the framework to only those platforms that the tool supports, and may also prevent end to end testing of tests that span applications and platforms.

## Logic

Error handling and recovery must be programmed into the window scripts as well as to the overall script driver. This also restricts the test developer’s control over options to respond to varying conditions.

## Class Engines

### Library Design

Class engines contain script functions that are specific to actions taken against a class of object. For example, entering or verifying the value of a text field. Tests are designed as sequences of individual steps against objects, and each step calls an action that is defined for the class of the object in use.

### Test Case Development

Class engines can use either spreadsheets or relational databases. If spreadsheets are used they may be difficult to manage and enforce naming conventions, but since each step follows the same format the columns are always consistent. Relational database implementations are even more powerful as they enforce structure and conventions, and the tables follow a standard layout because the test case structure is consistent.

### Application Specific or Portable

Unlike other engine types, class engines are not specific to the application. The script functions are mapped to standard class libraries and can be shared across applications. For example, an HTML class library can be used for any HTML Web application.

### Implementation and Maintenance Effort

As long as the application uses commercially available class libraries, the execution engine can be used “out of the box” without any additional development required. This means the framework an be licensed and there is no need for the customer to...
maintain in house scripting skills. Also, the functions are not affected by changes to the application; all application-specific information is stored as a map in the relational database. Best of all, maintenance can be mostly automated because all of the application components and test cases are cross-linked: instant impact analysis and global updates are available. Also, unless the class library changes—which is rare—the functions require no maintenance.

**Tool or Platform Dependent**

The script drivers for class engines can be written in a native, compiled language and the functions written in the tool or language of choice. Since each step calls a function, a single test can span different tools or languages and thus cross applications and platforms, seamlessly.

**Logic**

Just as the script drivers can be delivered in compiled code, so can error handling, recovery and test reporting be built-in to provide a complete infrastructure for error detection, reporting, handing. The test developer can also be offered a comprehensive set of flow control and error recovery options to decide how to manage execution and respond to errors without having to write code.

*Certiﬁ™ from WorkSoft, Inc. is currently the only commercially available framework that employs this approach. For more information about test frameworks or Worksoft Certify, call 214-261-9600 or visit www.worksoft.com*