CHAPTER 5

Functional Test Automation*

Functional testing assures that your implementation of SAP meets your business requirements. Given the highly configurable and tightly integrated nature of the SAP modules, as well as the probability that you will also integrate in-house applications or third-party plug-ins, it is a critical and challenging task requiring the verification of hundreds or even thousands of business processes and the rules that govern them.

This chapter explores the business case for automating your functional testing, the alternative automation approaches to consider, and organizational considerations and techniques for maintaining and managing your test automation assets.

WHY AUTOMATE?

Test automation is not a panacea, but it can make a dramatic difference in the quality and stability of your SAP deployment over the long term. The key is to understand when automation works and when it does not, and how to assure your success.

Business Case for Automation

There are three key benefits to automation:

1. Expand your test coverage.
2. Save time and resources.
3. Retain knowledge.

*This chapter was authored by Linda Hayes, CTO of WorkSoft, Inc.

Expanding your test coverage is one of the most valuable benefits of automation because it translates into higher quality and thus less costs associated with downtime, errors, and rework. Over the life of your SAP deployment you will likely experience an increase in the number of business processes it supports, either through the implementation of additional modules or integration with other systems.

As a result, each successive implementation or modification affects a greater number of business processes, which increases the risk and opportunity for failure. Even a 10 percent increase in total functionality still requires testing of 100 percent of the process inventory due to the risk of unexpected impact. The tightly integrated nature of SAP increases this risk.

As Exhibit 5.1 shows, a manual test process cannot keep pace with this expanding workload because time and resources available for testing are either fixed or even declining. In this exhibit, the lighter arrow indicates the processes that need to be tested and the dark arrow indicates the number of test resources. This combination of increasing processes that need to be tested with a static number of testers leads to increased risk and potential cost of failure.

Under the scenario represented in Exhibit 5.1, automation is the only practical answer. It enables one to capture tests as repeatable assets that can be executed for each successive release or deployment, so that the inventory of tests can keep pace with the inventory of business processes at risk.

This repeatability saves time and resources as well. Instead of requiring repetitive manual effort to reverify processes each time changes are introduced, tests can be automatically executed in an unattended mode. This allows your resources to focus on adding new

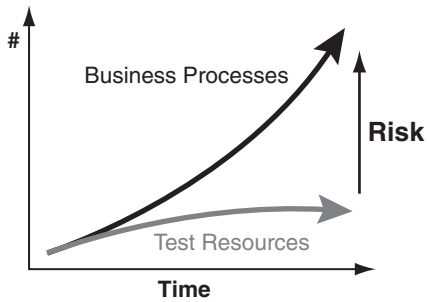


EXHIBIT 5.1 Test Workload Compared to Test Resources

tests to support new functionality instead of constantly repeating existing tests.

Ironically, when test time is short, testers will often sacrifice regression testing in favor of testing new features. The irony is that the greatest risk to the user is in the existing features, not the new ones. If a business process that the enterprise depends on stops working—or worse, starts doing the wrong thing—then you could halt operations. The loss of a new feature may be inconvenient or even embarrassing, but it is unlikely to be devastating.

This benefit will be lost if the automated tests are not designed to be maintainable as the application changes. If they either have to be rewritten or require significant modifications to be reused, you will keep starting over instead of building on prior efforts. Therefore, it is essential to adopt an approach to test library design that supports maintainability over the life of the application.

Finally, the process of automating your test cases introduces discipline and formality to testing, which results in the capture of application knowledge in the form of test assets. You cannot automate what is not defined. By defining your business processes and rules as test cases, you are converting the experience of subject matter experts (SMEs) and business analysts (BAs) into an asset that can be preserved and reused over the long term, protecting you from the inevitable loss of expertise due to turnover.

When to Automate

Conventional wisdom holds that you should automate your tests only for regression testing; that is, the initial deployment should be performed manually and only successive changes automated. This belief arises from the historical record/playback approach to test automation, which requires that the software be completed and stable before scripts can be captured.

New approaches exist, however, that allow automated tests to be developed well in advance of configuration or code completion. These approaches are further described later in the Test Automation Approaches section.

Using these new approaches, automated tests can serve a dual purpose: They can provide documentation of the "to be" business process as well as deliver test automation. This collapses two steps—

documentation and automation—into one, thus further conserving time and resources.

What to Automate

Automation is all about predictability. If you cannot express the precise inputs and expected outputs, you cannot automate a test. This means that it should be used to verify what is known or predicted. Typically this means positive tests, as in assuring that the business process is executed successfully, but can also be applied to negative tests that verify if business or field edit rules are violated, such as invalid data types or out-of-range values in which the data is rejected and an error message given. Think of these tests as "making sure" that processes work as expected.

In the context of SAP, the obvious automation candidates are the "to-be" processes, processes that are executed frequently, critical to the business, and contain integration points (touch points). For SAP-based production systems, SAP transaction ST03 allows for quick filtering of which SAP transaction codes are actually used in production and to what extent/volume.

Further, for each process, the data variations that exercise business and edit rules can also be automated. Applying data-driven techniques to automation enables you to quickly expand your test cases by adding data. This also means, however, that automation is not appropriate for ad hoc, random, or destructive testing. These tests must be performed manually because by their very nature they introduce unexpected or intentionally random conditions. Think of these tests as covering "what-if" scenarios.

Ad hoc tests are uniquely suited to manual testing because they require creativity and are deliberately unpredictable. By allowing automation to take care of what you expect to work, you can free your experts to try to break the system.

Critical Success Factors

Successful test automation requires:

- Management commitment
- Planning and training

- Dedicated resources
- Controlled test environment
- Pilot project

No project can succeed without management commitment, and automation is no exception. In order for management to manage, they must know where things stand and what to expect. By letting management know up front what investment you need to succeed with automation, then keeping them informed every step of the way, you can get their commitment and keep it. This requires a comprehensive project plan.

Your automation plan must clearly identify the total costs and benefits of the project up front, provide a detailed project plan with the required resources, timelines, and related activities, then track results and report to management regularly. Costs include selecting and licensing the right tool, training the team, establishing a test environment, developing your test library, and maintaining both the tests and the tool. The number and type of resources you need, the time required, and the specific activities will depend on the approach you adopt.

If and when obstacles are encountered, let management know right away. Get bad news out as early as possible and good news out as soon as you can back it up. Nothing is more disconcerting for management than to invest resources without seeing progress or, worse, by sudden surprises. Also keep focus on the fact that the test automation project will last as long as SAP is being used and maintained. Every successive release, update, patch, or new integration will need to be tested and the automated test assets accordingly maintained and reexecuted.

No matter how easy to use the tool is claimed to be, plan for training as well, and perhaps consulting. Learning a tool through trial and error is costly and time consuming, and it is better to get off on the right foot. Since it is easier to get money allocated all at once instead of piecemeal, be careful not to buy the software first and then decide later you need training or additional services.

Although the promise of automation is exciting, realize that test tools do not work by themselves. Buying a test tool is like buying a treadmill—the only weight you lose is in your wallet! You must use the equipment, do the exercises, and sweat it out to get the benefits. Also understand that even though test automation saves time and resources in the long run, in the short term it will require more than

manual testing. Make sure management understands this, or you may find yourself with a tool and no one to implement it.

Not only must you have the right resources, you must also commit to a controlled test environment that supports predictable data. Automation is all about repeatability, and you cannot repeat the same tests if the data keeps changing. In most cases the data values are the key to the expected results. Identifying, creating, and maintaining the proper data is not a trivial problem to address and often represents more than half of the overall effort. Do not wait until you are ready to start testing to implement your strategy.

The ideal test data strategy is to have a known data state that can be archived and refreshed for each test cycle. If this is not possible or practical, you may consider using automation to "seed" or condition the data to create or modify data to meet your needs.

If this is your first automation effort, start with a small pilot project to test your project plan assumptions. Invest two to four weeks and a couple of resources in automating a representative subset of your business processes, and carefully document the effort and results during the pilot as these results can be used to estimate a larger implementation. Since you can be sure you do not know what you do not know, it is better to learn your lessons on a small scale.

Also be sure to commit the right type of resources. As described in the following section on test automation approaches, depending on the approach you adopt you will need a mix of skills that may or may not be part of your existing test group. Do not imagine that having a tool means you can get by with less skill or knowledge: The truth is exactly the opposite.

Common Mistakes

Pitfalls to avoid when automating your SAP testing include:

- Selecting the wrong tool.
- Using record and play techniques.
- Writing programs to test programs.
- Ignoring standards and conventions.

In order to select the right test tool you must perform an evaluation in your environment with your team. This is the only way to

assure that the tool is compatible with your SAP implementation—including any gap applications—and especially that your team has the right skill set to be productive with it. A scripting tool that requires programming skills, for example, will not be successful unless you have technical resources available on your team.

For purposes of this evaluation, make sure you understand how the tool handles not only test development but also test management and especially maintenance, since these are critical to long-term productivity. Do not settle for a simplistic record-and-play script. Insist on understanding how to write robust tests that are well structured, documented, reliable, and easy to maintain.

Record and play is a very attractive approach: Simply perform a manual process and have it automatically recorded into a script. But while these scripts are easy to record, they are unstable when executed and all but impossible to maintain. They do not have the documentation and structure to be readable, and they lack any logic to detect and respond to the inevitable errors and changes that will occur. Even variations in the response time of an SAP transaction can cause failures.

Another drawback to recorded scripts is that they contain hard-coded data. Recording the process of creating a hundred invoices, for example, will yield a script containing the same steps 100 times over. This means if a configuration change is made to any step of the process, it must be made 100 times. Since this is impractical, recorded scripts are rarely reused after changes and must often be re-recorded. Thus, the value of automation is lost.

While the issues with capture/playback can be resolved using advanced scripting code, this leads to the other extreme: writing programs to test programs. This technique requires programming skills, which may exclude your most experienced testers. Further, if each test case is converted to script code, you will have more code than the SAP module does. This approach results in custom code that is also difficult to maintain, especially by anyone other than the original author.

Balancing the trade-offs between ease of use and coding is the subject of the discussion of test automation approaches in the next section.

The last common mistake is to ignore the need for naming standards and test case conventions. If each tester is permitted to adopt their own approach and store their tests wherever they wish, it will

be impossible to implement a centralized, unified test library where tests can be easily located and shared. Treat your automated tests as the asset they are and ensure that they are easily found, understood, and maintained.

TEST AUTOMATION APPROACHES

Test automation has steadily evolved over the past two decades (longer if you count mainframes) from record and play, which is all code and no data, to code-free approaches that are all data and little or no code. This trend reflects the fact that code is more costly to develop and maintain than data.

This evolution has followed these four stages:

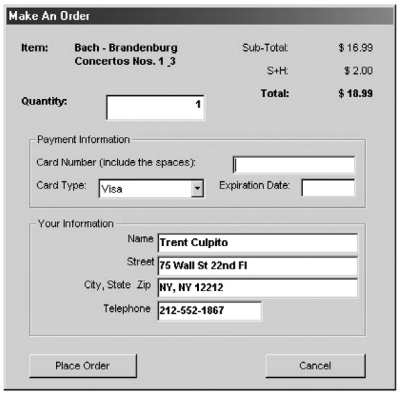
1. Record and play
2. Data-driven
3. Frameworks
4. Code-free automation

These represent varying combinations of code and data used to construct test cases and each has different advantages and drawbacks.

Record and Play

Record and play appears to be easy but turns out to be difficult. Recorded scripts usually have a very short useful life because they are hard to read, unstable when executed, and almost impossible to maintain. The time that is saved during the initial development is more than offset by the downstream costs of debugging failed sessions or re-recording after changes. Exhibit 5.2 shows an example of a recorded script.

The ideal use of record and play, oddly enough, is to capture the results of manually executed tests. This assists the tester in documenting results and perhaps reproducing the exact steps that led to any errors.




```
'Test Case Script

Sub Main

    'Make An Order
    Window SetContext, "Name=frmOrder", ""

    'Card Number
    EditBox Click, "Name=txtCreditCard", "Coords=16,9"
    InputKeys "3333444455556666"

    'Expiration Date
    EditBox Click, "Name=txtExpirationDate", "Coords=6,7"
    InputKeys "12/2005"

    'Place Order
    PushButton Click, "Name=cmdOrder"

    'Confirmation Screen
    Window SetContext, "Name=frmConfirm", ""
    PushButton Click, "Name=cmdOK"

End Sub
```

EXHIBIT 5.2 Example of Recorded Script

Traditional test automation tools that cost thousands of dollars per user are overkill for this use. Instead, look for simple session recorders that are available for as low as $100.

Data-Driven

Data-driven techniques address the hard-coded data issue of record and play by removing the data from the scripts and instead reading it from an external file. Typically, a process is recorded once, then script code is added to substitute variables for literal data, read the variable values from a file, and loop until all records are completed.

This approach reduces overall code volume and allows test cases to be added quickly as data records, but requires programming skills to implement. It also results in custom code for each process that must be maintained as the application changes. Exhibit 5.3 reflects the type of changes introduced into a recorded script in order to make it data-driven.

Frameworks

While data-driven techniques succeeded in reducing code volume attributable to hard-coded data, they did not directly address the

```
'Data Driven Framework
'Test Case Script

'$Include "SQAUTIL.SBH"

Sub Main
    Dim Result As Integer
    Dim DatapoolHandle As Long
    Dim DatapoolReturnValue As Variant

    'Open the datapool
    DatapoolHandle = SQADatapoolOpen("OrderFormDP")
    '...Add error checking....

    'Loop through the datapool
    While SQADatapoolFetch(DatapoolHandle) = sqaDpSuccess
        'Open Order Form
        Window SetContext, "Name=frmMain", ""
        PushButton Click, "Name=cmdOrder"
        Window SetContext, "Name=frmOrder", ""

        'Card Number
        Result = SQADatapoolValue(DatapoolHandle, "Credit Card Number", DatapoolReturnValue)
        '...Add error checking....
        EditBox Click, "Name=txtCreditCard", "Coords=16,9"
        '...Clear Value...
        InputKeys DatapoolReturnValue

        'Expiration Date
        Result = SQADatapoolValue(DatapoolHandle, "Expiration Date", DatapoolReturnValue)
        '...Add error checking....
        '...Clear Value...
        EditBox Click, "Name=txtExpirationDate", "Coords=6,7"
        InputKeys DatapoolReturnValue

        'Place Order
        Result = SQADatapoolValue(DatapoolHandle, "Order", DatapoolReturnValue)
        If UCase(DatapoolReturnValue) = "YES" Then
            PushButton Click, "Name=cmdOrder"

            'Confirmation Screen
            Window SetContext, "Name=frmConfirm", ""
            PushButton Click, "Name=cmdOK"
        Else
            PushButton Click, "Name=cmdCancel"
        End If
    Wend 'Go fetch next row

    'Close datapool
    Result = SQADatapoolClose(DatapoolHandle)
    '...Add error checking....
End Sub
```

Edit Datapool - OrderFormDP

	Credit Card Number	Expiration Date	Order
	1111222233334444	12/2005	Yes
	1111222233334444	1/2005	Yes
	1111222233334444	13/2005	Yes
▶	1111222233334444	0/2005	Yes
*			

EXHIBIT 5.3 Example of Data-Driven Script and Data File

inefficiencies of not sharing common code to handle common tasks across test cases. They also limited the analyst's ability to design test flows consisting of multiple scenarios and data.

In response, frameworks evolved as a way to provide an infrastructure to handle common tasks and allow business and quality analysts to write test flows by calling reusable code components.

Typical elements of a framework include:

- A layer that allows test flows to be constructed as data in a spreadsheet or database.
- Reusable or generated code components that execute testing tasks against SAP.
- An infrastructure that handles test packaging, timing synchronization, error handling, context recovery, result and error logging, and other common tasks.

Frameworks require two roles and skills: the test engineer, a programmer or scripter who develops the framework and reusable code components, and the test designer, a business or quality analyst who constructs processes by ordering these components, together with the test data values they require, into a spreadsheet or database.

A framework offers several advantages. Nontechnical testers can design automated test flows and provide data in a standard format, and test engineers can optimize their coding and maintenance effort by developing reusable components. The framework also takes care of managing and monitoring execution to provide reliable results.

There are three basic types of frameworks: key/action word, screen/window, and class library. Each type can be implemented using text files (spreadsheets) or databases. Spreadsheets are more economical, as most users already have access to them and are familiar with their use, but they are more challenging to manage and maintain because they are not centrally stored or controlled. It is also easier to make typographical or spelling errors in a spreadsheet.

A database, however, requires more cost and effort to implement but is easier to manage. By providing a graphical user interface (GUI) front end, users can select from drop-down lists and otherwise be protected from making input errors. Relational databases also enable more rapid maintenance as mass changes can be introduced using Structured Query Language (SQL) statements and similar functions.

Key/Action Word Framework A key or action word framework comprises business functions that perform tasks against SAP such as entering an order or verifying an invoice. Each key or action word has a set of data values associated with it for either input or verification. Exhibit 5.4 illustrates a typical key/action word implementation using spreadsheets.

Key/action word frameworks can be developed internally or acquired from commercial vendors. Some of the commercial tools generate the scripts for common components, then allow test engineers to add additional code to handle errors and decision making at runtime as well as other application-specific logic or functionality.

The maintenance of key/action word frameworks is divided between the code and the data. The code may have to be regenerated or modified when the application behavior changes and the spreadsheet or database may have to be updated as functionality is enhanced or changed.

Screen/Window This type of framework is a variation of key/action word in that there are reusable code components that perform specific tasks, but in this case they are organized around actions such as data

EXHIBIT 5.4 Key/Action Word Implementation Using Spreadsheets

Test Name:	**Add Order**			
Description	Create orders and verify total and tax			
Testcase	**Customer**	**Product**	**Quantity**	**Price**
Add Order	Acme Signs	Posterboard	1000	5
Add Order	*Baltimore Sun*	Paper	65000	1.15
Add Order	Crosstown, Inc.	Confetti	1250000	0.05
Testcase	**Customer**	**Product**	**Tax**	**Total**
Verify Order	Acme Signs	Posterboard	400	5400
Verify Order	*Baltimore Sun*	Paper	5980	80730
Verify Order	Crosstown, Inc.	Confetti	0	1000000

input or verification against each SAP screen. Exhibit 5.5 shows a screen/window implementation using a database and GUI interface.

When a screen changes, the related screen action code components must be modified or regenerated as well as the related test case spreadsheet or database.

Class Library A class library framework is built around code components that are mapped to SAP objects instead of tasks or screens. Each object class has an associated set of actions that can be performed against it—for example, input to a text box or pressing a button. These class/action code components may be developed or generated, with code added for decision-making logic based on the results during execution. Exhibit 5.6 is an example of a spreadsheet implementation for a class library framework.

As with other framework types, these can be organized into test processes in spreadsheets or databases. In this case, the data is provided for each single action.

Since the SAP class library rarely changes, the only code that requires maintenance for functional changes is any decision-making or other custom code that has been added. The rest of the maintenance occurs in the spreadsheets or database.

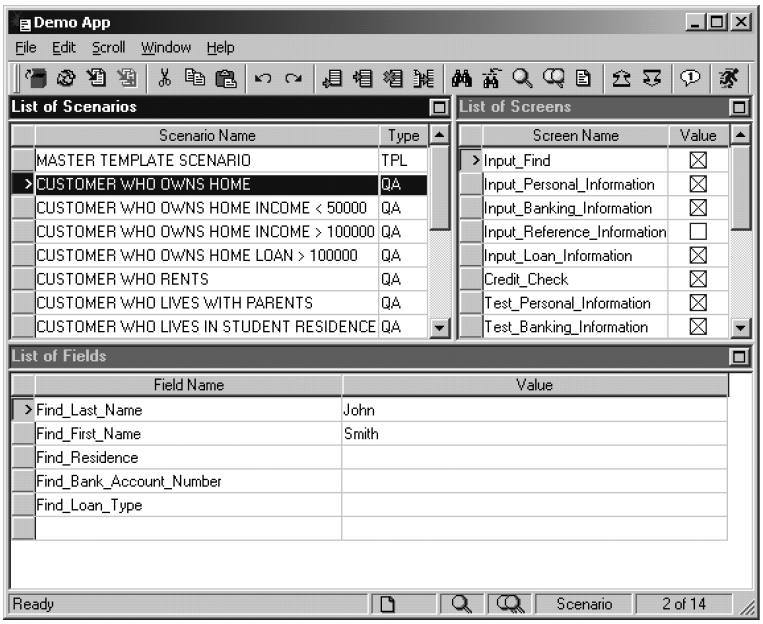


EXHIBIT 5.5 Screen/Window Implementation Using a Database and GUI Interface

Build versus Buy Any of these framework types can be internally developed or licensed from a commercial vendor. While building your own framework may sometimes appear to be less costly and provide the most flexibility and control, it requires an investment in the development and ongoing support and maintenance of the framework. Since robust frameworks consist of tens of thousands of lines of code, the

Test Case	Window	Object	Class	Action	Data
Add Order	Logon	User ID	TextBox	Input	Test User
	Logon	Password	Password	Input	*****
	Logon	Submit	Button	Press	
	Order Entry	Customer	TextBox	Input	Acme Signs
	Order Entry	Product ID	ListBox	Select	Posterboard
	Order Entry	Quantity	TextBox	Input	1000.00
	Order Entry	Price	TextBox	Verify	5.00
	Order Entry	Submit	Button	Press	

EXHIBIT 5.6 Spreadsheet Implementation for a Class Library Framework

resource costs and time to create and support this code may be substantial.

Further, if the original framework developers leave, it is common for the replacement engineer to rewrite or restructure the framework code according to their own style or preferences. This adds to the ongoing cost of ownership.

Of course, buying a framework incurs a licensing fee, but this cost may be offset by reducing the continued support and maintenance costs to a fixed-price annual fee. The decision as to which option is more economical should also take into consideration how much custom code is needed in either scenario. If the commercial framework still needs significant code development to support the desired test work flows, it may not offer enough of a cost advantage to offset the license costs.

Code-Free Automation

A new type of test automation solution has recently emerged that does not require any code to be developed at all. This approach includes vendor-supported reusable code components that are mapped to the SAP class library and allows test analysts to construct processes using point and click within a GUI interface. The tester selects the SAP screen, the object and the action to be performed from a series of drop-down lists, then provides the test data or variable name for any required values.

The difference between the code-free approach and the previous frameworks is that no code is written or generated in order to automate the tests. All test processes are stored as data within a database. Even decision making is supported through a GUI without requiring the development of any additional code.

In this approach, the application screens and fields are defined either by learning the SAP screens or by extracting the screen information directly from the SAP metadata. This information is stored as a map within the database and it is used to populate the drop-down lists as tests are defined. Test analysts can further select from predefined options for making decisions at runtime to control the process workflow. Exhibit 5.7 depicts an example GUI process editor for a code-free automation solution.

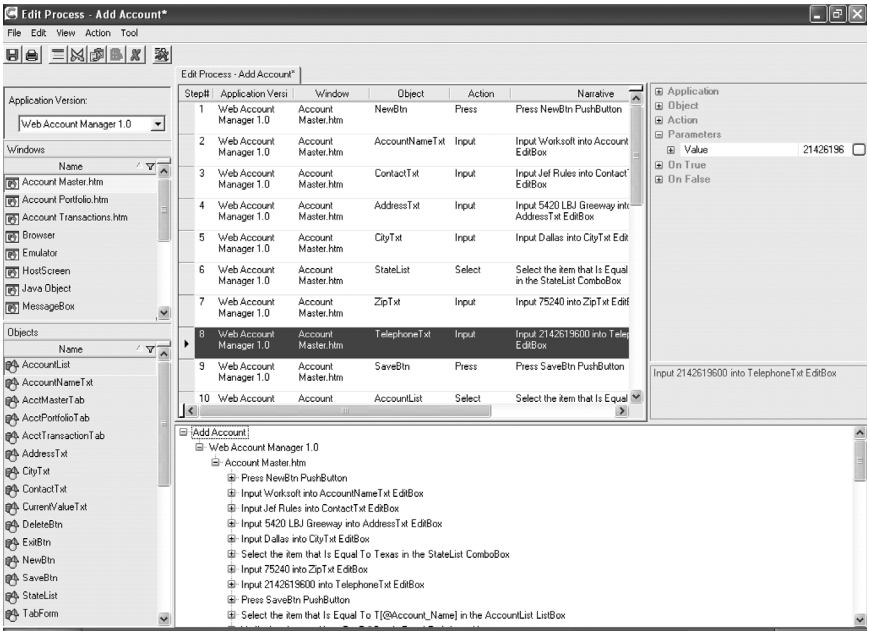


EXHIBIT 5.7 Example GUI Process Editor for a Code-Free Automation Solution

Aside from removing the need for test engineers to develop and maintain custom code, code-free solutions enable automated maintenance. As the application changes, the map components are compared and all differences documented, including the affected test assets. Global changes can be made automatically as well to modify or remove test steps related to changes or deletions. Since all test assets are stored as data, this can be more easily accomplished than finding impact and making changes to code.

Even a code-free solution, however, should support extensibility options in the event that your implementation of SAP contains interfaces to non-SAP applications to fill gaps.

TEST LIBRARY MAINTENANCE

The primary benefit of automating your SAP test processes is for future reuse as configuration changes are made or new patches or

versions installed. By automatically reexecuting all of your test processes after changes, you can ensure that there have been no unintended effects. This level of test coverage can prevent costly production errors, failures, and downtime.

In order to enjoy this benefit you must be able to maintain your test processes as changes are made to SAP or your configuration. If you do not update the tests each time you make a change, they will become obsolete. In the same vein, you must add new test processes or test data to verify new functionality as it is added so that your test coverage continues to expand as your usage of SAP does.

One way to limit maintenance time and overhead is to adopt a framework or code-free approach so that script code maintenance is limited or eliminated entirely and most changes occur in data instead.

Because maintenance is an ongoing requirement, it is critical that it be efficient. Extensive manual changes to custom-coded components may be too time-consuming or difficult, resulting in a reduced useful life for your automated tests. This means you must design your tests to be easily maintained by following development standards and naming conventions, and by enforcing change management and version control on all test assets.

Maintenance Events

There are three primary events that can trigger maintenance of your test assets. The first arises when your SAP configuration changes, whether to modify screens or the business process workflow. Depending on your automation approach, this will require that your test components—whether stored in code, spreadsheets, or a database—be modified to accommodate the differences.

The second maintenance event is a change to a business process due to new or different rules. The SAP screens themselves may not be modified, but the rules that govern the workflow may be changed. In a script code–based framework, this may necessitate scripting or regeneration of code; a code-free solution will need only changes to the test processes.

Changes to data can cause the third type of maintenance event. This change may arise from different data in the test environment it-

self, or new data may be needed to exercise new process or rules. Unless you are using record and play, your test data should be located in a text file, spreadsheet, or database.

In each of these cases it is important that your naming conventions or coding standards permit you to easily identify which test assets are affected by changes without individually inspecting every test process or data file. Depending on the automation technique and framework type you select, the impact of a change may be analyzed automatically or manually. Generally, assets stored as code make it more difficult to locate and make changes than assets stored as data. Similarly, data housed in a database is easier to manage and maintain than data stored in text files or spreadsheets.

Version Control

Because maintenance events result in changes to test assets, it is necessary to institute version control. Prior versions of a test should be kept in case the functionality has to be rolled back, or for audit trail purposes to comply with regulatory requirements.

If your tests are stored as script code, you may use a software source control system that supports check in/check out for code modules and allows you to identify differences and perform merges between versions.

If your tests are stored as data in text files or spreadsheets, you may also use most software source control systems. For test assets stored in a database, make sure the database schema permits multiple versions to be maintained and compared, and if a test asset is being modified, that it is protected from being overwritten by someone else.

MANAGING TEST AUTOMATION

Your test automation team will require a mix of skills, depending on the approach you have selected. Estimating the time and effort will also depend on the techniques and tools you have adopted.

Team Roles

As described in previous sections, the code-based approaches and frameworks require a minimum of two roles: test engineers, who develop and maintain the script code components, and test analysts or designers, who construct and execute the test processes and data.

Test designers should be SMEs or BAs who have domain expertise in the business processes to be tested. Test engineers need to have programming skills and either training or experience with the scripting tool of choice. Test analysts need SAP domain expertise and business process experience. If you have adopted a database repository, you will also need database administration skills.

Whether your test framework is internally developed or commercially licensed, you will need to plan for training the test team on how to design and develop reusable, maintainable test processes.

It is important not to skimp on training team members on naming standards and coding conventions. These are essential skills for implementing a test library that can be managed, maintained, and transferred over the long term.

Estimating

Estimating the timeline for your test automation effort requires you to consider the following factors: the automation approach you adopt, the number of business processes to be executed, and the number of business rules to be verified.

For example, if you select the key/action word framework approach you will need to define the inventory of key or action words that are needed, together with any custom decision-making code. Generally, if it takes one hour to record a process, it will take another five to modify the script to add variables, timing, logic, error handling, and so forth, plus another five to integrate it into the framework, test, and debug it. So a one-hour manual process will take about 10 hours to reduce to a script code component. From there, additional rules can be tested by adding rows of test data, which may be rapid if the data is already defined and slower if not.

If you are developing the framework internally, add time to develop and test the infrastructure as well. A typical custom framework

infrastructure for a single application is about 50,000 lines of code. Plan for time to design the library, develop, test, and document it. At 25 to 50 tested lines of code (LOC) per developer per day, this translates into about four to eight person-years of development.

Likewise, the screen/window approach can be estimated by counting the number of SAP screens you need to traverse, then multiplying by the number of actions you intend to support for each (e.g., input, verify, and store). Finally, automate one screen of average complexity and use it as a baseline to project the remaining effort.

The class library implementation can be estimated by identifying the number of classes and related actions plus the infrastructure. There are about 12 different GUI object classes in the SAP GUI; if you provide an average of five actions for each one of approximately 50 LOC each, you will have 600 LOC for the classes and actions plus any custom code needed.

After that, estimate the number of test processes and test data values needed; developing the test workflows may take from half an hour to an hour including writing, testing, and debugging. Adding test data to a workflow to exercise different rules may take only a few minutes by adding rows to a data file.

Code-free approaches require estimates for the number of business processes and rules to be verified. Processes can typically be constructed in 15 minutes to half an hour depending on complexity, and test data can be added in minutes as another row in a table. This does not include any extensions for non-SAP applications.

In all approaches, however, be certain to plan for gathering and analyzing the business process flows and the business rules and related data. Ideally, these were documented during the initial business process engineering phase in the form of the "to-be" processes. If not, plan time to interview application subject matter experts to extract this information. Exhibit 5.8 summarizes the estimating factors for each approach.

OUTSOURCING SCRIPTING TASKS

If you adopt one of the techniques that requires test engineers—and especially if you elect to build instead of buy your framework—your organization will need skilled script coders. If you do not already have

EXHIBIT 5.8 Effort Estimation Factors by Approach

Approach	Framework	Code Components	Data Components
Key/action word	50,000 LOC 25–50 LOC/day or licensed	# business tasks × 200 LOC each	# processes × test case variations × 1 minute per row
Screen word	50,000 LOC 25–50 LOC/day or licensed	# screens × 4 tasks × 100 LOC each	# processes × test case variations × 1 minute per row
Class library	50,000 LOC 25–50 LOC/day or licensed	10 classes × 5 actions × 50 LOC each or licensed	# processes × number of steps × 30 seconds per step plus # test case variations per process × 1 minute per row
Code-free	Licensed	Licensed	# processes × number of steps × 30 seconds per step plus # test case variations per process × 1 minute per row

these resources available, you have three options: Hire new employees, retain contractors, or outsource.

Outsourcing may offer the benefits of reduced costs and access to resources already skilled in the test tool at hand. However, realize that the test designer role requires domain expertise and cannot be outsourced.

The biggest challenge of outsourcing is facilitating efficient communication and project management between the designers and engineers, especially if the engineers are offshore. Be sure to include extra time for detailed, explicit test case documentation to support remote engineering. Insist on industry best coding practices such as naming standards, coding conventions, version control, and documentation, as discussed previously in this chapter: All are essential to assure the long-term viability of your automated tests.

Finally, plan for the results to be reviewed and analyzed by the designers since they are the owners of the processes and ultimately accountable for their accuracy.

SUMMARY

Test automation is a strategic solution to assuring that your SAP implementation is accurate and reliable both the first time it goes live and after every other time that configuration or software changes are made. Thorough, automated test coverage can save millions in production errors, downtime, and loss of user productivity by detecting issues before they impact the business.

So take the time to select the right tool and technique for your needs, invest the proper resources, and follow best practices so that your test automation library can serve as a long-term asset.