

## White Paper



### Application Deployment in the Securities Industry: Making Functional Test Automation Work

---

Debashis Pradhan, Eric Librea and Kanna Venkatasamy

Businesses that need to test complex interactions of systems often find that their testing capabilities are not as robust as they would want them to be. Using a rigorous approach to clearly define the boundaries of systems that need to be tested, optimizing the test cases to a manageable number and executing them effectively would provide significant benefits in the short and long term.

Automation has always been viewed as an unavoidable means to achieve the desired amount of rigor and consistency in the testing process. Through automation, system development teams are relieved of the error-prone and tedious tasks related to creating test plans, defining and executing test cases and scenarios, and measuring and evaluating test results. Advancements in the automated testing arena have helped many project teams increase the quality of their work and, consequently, produce tangible business benefits to the enterprise.

With systems operating in a tightly interconnected environment in the financial services world, the criticality of automation increases exponentially because of several reasons:

- Applications are frequently modified and enhanced in short release cycles due to ever changing customer and regulatory needs
- Cost of failure is higher as the success of running one system is dependent on the success of running all other systems
- Volume of test cases is larger because of the increased variability in input data resulting from the very nature of the financial services processes and the chain of interconnected systems

Across a variety of client organizations, we have observed common shortcomings in their automated testing capability. These shortcomings are a result of both process and technology limitations that we believe would negatively impact the achievement of benefits from automated testing. These shortcomings are:

- **SCOPE OF TESTING:** Testing, though geared towards end-to-end testing of a network of applications, is focused primarily on the main system
- **MODE OF TESTING:** Testing is typically dependent on events initiated by the GUI front-end of the system being tested
- **EXTENT OF TESTING:** Testing is usually based on a limited and sub-optimal number of test cases

This viewpoint discusses Infosys position on these three observations and suggests the required process and system improvements to overcome these shortcomings.

## Scope of testing: Looking beyond the boundaries

Testing tools and methods that are effective in testing the system under change are often limited when it comes to testing the complete impact on downstream systems.

Systems are becoming increasingly interconnected and often a small change in one system requires rigorous testing in some other downstream or upstream system. There are several trends that give rise to such a situation:

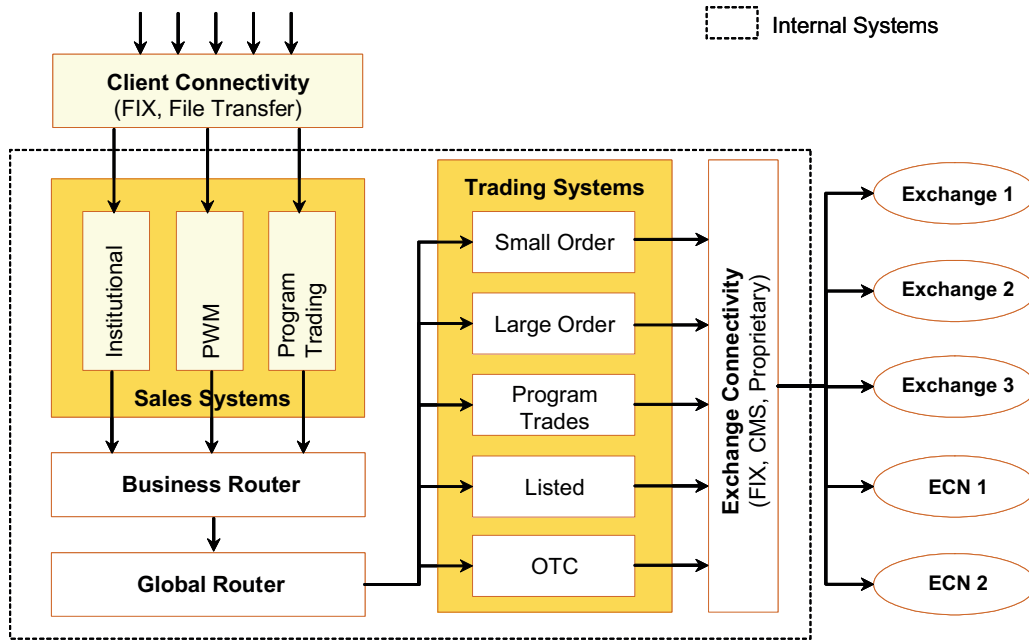
- Establishing shared services such as CRM, data warehouse, billing and enterprise components that are used by many business groups and systems
- Increased use of middleware and workflow technologies where the processing logic is spread over a set of systems
- Increased adoption of services oriented architecture and Business process management techniques that focus on abstracting meaningful business components and reusing them
- Increased adoption of packaged and best-of-breed solutions necessitating increased integration

Testing these interdependencies effectively is even more critical for core applications. This can be observed in the securities industry, where Order Management Systems (OMS) processing huge volumes of orders interface with a variety of front-end customer systems and back-end trade processing, exchange and settlement systems. Additionally, there are multiple implementations of OMS across lines of business or geographic regions. Changes in OMS need to be rigorously tested to avoid orders not being processed in downstream systems, a scenario that could lead to serious losses.

In the realm of regression testing, the focus is to ensure that the original functionality is not disturbed when new functionality or changes are made to a system. This often leads to regression testing in isolation where the interfaces to other systems are simulated. This assumes that the other interconnected systems are not undergoing changes and also assumes that the interconnections have been tested well already. End to end testing of such complex functionality flows is costly and hence the tests were often limited to basic conditions.

Ability to perform end to end testing efficiently across a network of systems requires different modeling techniques, approach and capabilities that are often different from the traditional tools and methods that focus on regression testing with-in a particular system. Transaction flow modeling is one well-known technique that can be used to assist in defining the scope of testing. It depicts the interactions within a network of applications. The transaction flow model uses nodes and links to represent cohesive logical processing steps and their order of occurrence. The flow diagram is used to visualize the network of applications and arrive at inputs to the test case generation technique discussed later in the article.

## EXHIBIT 1 – Application Interactions (Order Management Example)



Source: Infosys Analysis

### Mode of testing: Debunking the myth of GUI record/playback

Most QA teams in securities firms use test automation tools that use a mix of record/playback and scripting approach for functional testing. However, the expected ease of use and ROI from these tools remains elusive as these firms grapple with creating and maintaining test scripts in an ever changing environment. Over a period of time these tools either gather dust or are supplemented by significant manual testing.

Let us take a closer look at some of these challenges.

#### *Applications are complex*

These applications automate numerous processes and handle multiple product lines across geographic regions, have multiple flow paths and handle vast amounts of information in real-time. As such these systems are not amenable to traditional test automation. While record/playback is useful in cases where the focus of the testing is on the user interface, it is not the best approach when the core application logic needs to be tested.

Consider the example of a trading screen user interface. It has about 15-20 different parameters on the GUI but there would be many more that are processed by the application logic but are not captured at the user interface level. These additional fields could be:

- **Enrichments**  
For instance, an Account Name entered on the screen will pull in the Account Number, Type, Alias, Corresponding Sales Rep information. Thus one Account field on the user interface translates to perhaps 12-15 parameters.
- **Status Fields**  
Parameters like Transaction Types, Dates, Times, Tags, Unique Identifiers etc. are added at the application server level, something which the GUI is unaware of.
- **Placeholders**  
Many parameters are added for future use, Split Quantity for instance, which gets populated if an order is split, or Executed Quantity which gets populated when an order is executed.

As is obvious, GUI testing based on record/playback would be insufficient to comprehensively test all these nuances. For instance, how do we test the application logic's ability to handle negative values when there is GUI level validation? Or, how does the server behave if a new order is sent with a non-zero executed quantity?

## Applications change frequently

While creating test cases is relatively easy and straightforward, there are challenges in executing these test cases. For example, tests would have to be re-recorded whenever the application undergoes changes. Moreover, tests could become unreliable due to data variability.

While record / playback offers a stop-gap solution, using this approach for creating entire scripts for an application will make the scripts difficult to maintain when there is a change in the application.

## Creating and maintaining scripts is intractable

Contrary to popular belief, scripting is neither easy nor cheap. "It usually takes between 3 and 10 times as long to create, verify, and minimally document the automated test as it takes to create and run the test once by hand"<sup>1</sup>. Scripting tools in addition to providing record/playback capabilities, offer a programming interface. While this allows for more control and addresses the problem of frequent changes to the application and test data, the skill and effort requirements are more than what it is made out to be. Our engagement experience shows QA teams face an uphill task in maintaining and executing numerous scripts created over a period of time while using traditional test automation tools. Not only do they face challenges in ensuring high code coverage, they also face the prospect of having to analyze reams of test reports. Driven by cost and time pressures, most QA shops automate only a slice of their test cases and seldom maintain test assets which eventually lose currency.

For QA teams faced with these problems there is a way out. Using concepts of functional decomposition and test asset reuse, test teams can significantly assuage their problems.

Functional decomposition involves reducing all functions and processes handled by the application to their lowest unit of work or process. Each of these processes is defined by a set of parameters or attributes. Test cases are then created using these atomic level processes or tasks. A list of such processes is grouped and a sequence of execution specified to create a test scenario. These test cases or scenarios are then run directly against the application logic.

To ensure test case reuse, test data is separated from the test case itself. In other words, we define the processes that form a test case, but the values that parameters for these processes can take up is kept separate.

The advantages of using such an approach are manifold:

- Test case reuse significantly brings down the effort spent on maintaining scripts
- Separating the test data from the test itself eliminates redundancy and duplication in scripting
- Multiple test cases can be grouped logically to form complex test scenarios or test suites (e.g. test cases related to new order, order cancel, order replace, etc can be grouped to form an "Order flow" scenario)

## Extent of testing: Too many is not enough

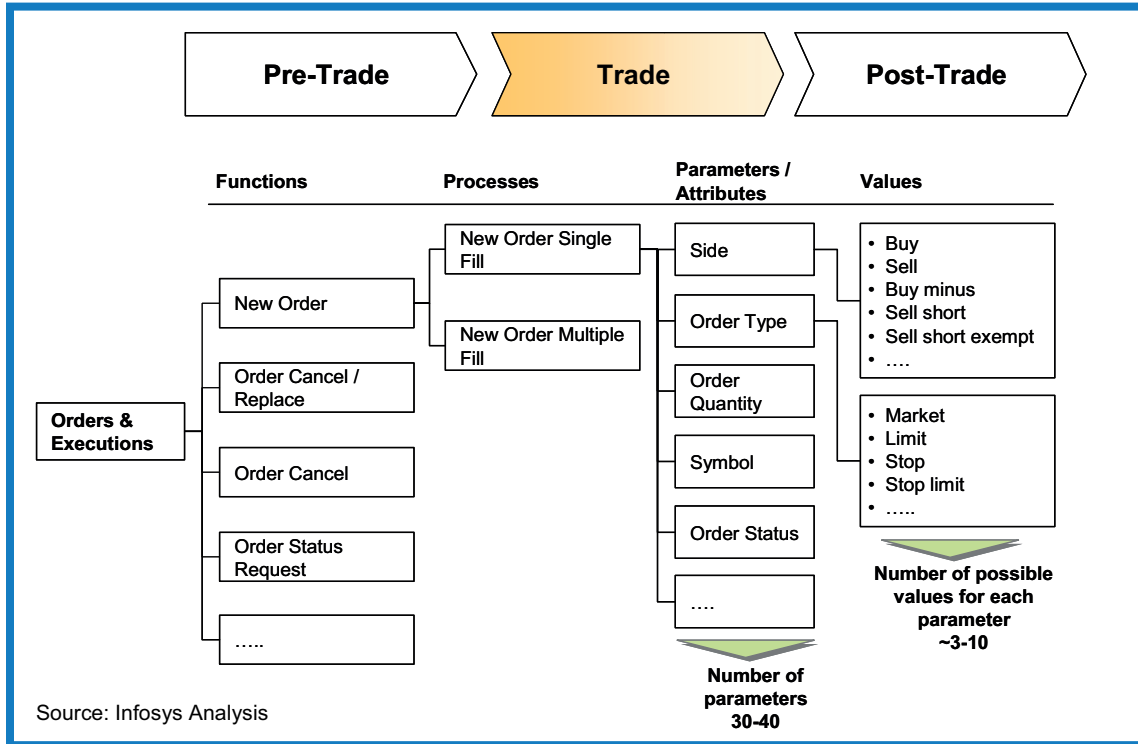
The financial services industry, particularly the securities industry is characterized by complex, business-rules driven applications that support order management, trade processing, custody and fund accounting, etc. These systems typically handle several processes, each process defined by multiple parameters or attribute values. Adding to the complexity is their interface with upstream and downstream applications creating an interconnected world of dependent processes and systems. In such a scenario, the number of possible combinations to functionally test the application easily runs into millions.

Let's take the example of an Order Management System (OMS) in a securities firm. Even if we consider just one trade order-related process i.e. new order with single fill, there will be several parameters or attributes describing such a process. Each of these parameters, [say order type](#), can take up several possible values like market, limit or stop. In a real-life situation there would possibly be more than 30-40 parameters describing an order, with 3-10 possible values for each parameter.

<sup>1</sup> Cem Kaner, "Improving the Maintainability of Automated Test Suites"

To ensure that automated testing is rigorous, the QA team needs to have test cases that are exhaustive and comprehensive in nature. Doing so requires taking all external inputs to the system as independent parameters and using these to generate test cases. It can be readily estimated in our example that the number of possible combinations of test cases is in the high order of magnitude ( $\sim 3^{30}$  to  $10^{40}$ ). And this is without even considering other processes or applications.

## EXHIBIT 2 - A Multitude of Combinations



In practice, however, the set of all possible combinations is too large to explore and it becomes impractical to run automated tests using this large number of test cases because of cost and time considerations. As a result, QA teams often err on the side of defining a limited number of test cases (usually based on hunch and gut feel) that do not fully account for all the input variations. Needless to say, the chances of encountering a gremlin in the production environment are very high.

A more thorough approach to cover all critical cases, while keeping the number of test cases under control, would be to implement the notion of pair-wise coverage<sup>2</sup>. The basic premise of pair-wise coverage is that a defect is usually found to involve a single condition or two conditions, variables or attributes. As an example, a car might not brake properly either due to worn brakes (single condition) or the concurrence of worn brakes and a leak in the brake line (two conditions).

Complete testing through all possible combinations is only necessary when a defect depends upon every parameter or attribute in the system to be in one specific state. It's like saying the car didn't brake properly because of worn brakes, a leak in the brake line, engine misfire, worn tires, wheel misalignment, all happening at the same time, but independently not sufficient to cause the problem. While there is no denying that such cases do occur, they are far too rare to merit attention. Several recent studies indicate that the code coverage using a pair-wise approach can be as high as 90-95 percent<sup>3</sup>.

<sup>2</sup> Siddhartha R. Dalal et al., "An Approach to Testing Based on Combinatorial Design", July 1997

<sup>3</sup> Siddhartha R. Dalal et al., "The Combinatorial Design Approach to Automatic Test Generation", Sept 1996

How can QA teams maximize their chances of finding a problem in the limited time available? Test cases are optimized when the team generates the minimal number of test cases such that every value of every parameter is tested at least once with every other value of every other parameter. Let's draw the analogy with Order Management.

Consider three order-related parameters and each parameter with two possible values (refer Exhibit 3). The number of all possible combinations is 8.

### EXHIBIT 3 – Winnowing out the duplicates

Order Attribute	Possible Values
Side	<ul style="list-style-type: none"> <li>Buy</li> <li>Sell</li> </ul>
Type	<ul style="list-style-type: none"> <li>Market</li> <li>Limit</li> </ul>
Time in Force	<ul style="list-style-type: none"> <li>Day</li> <li>Good till Cancel</li> </ul>

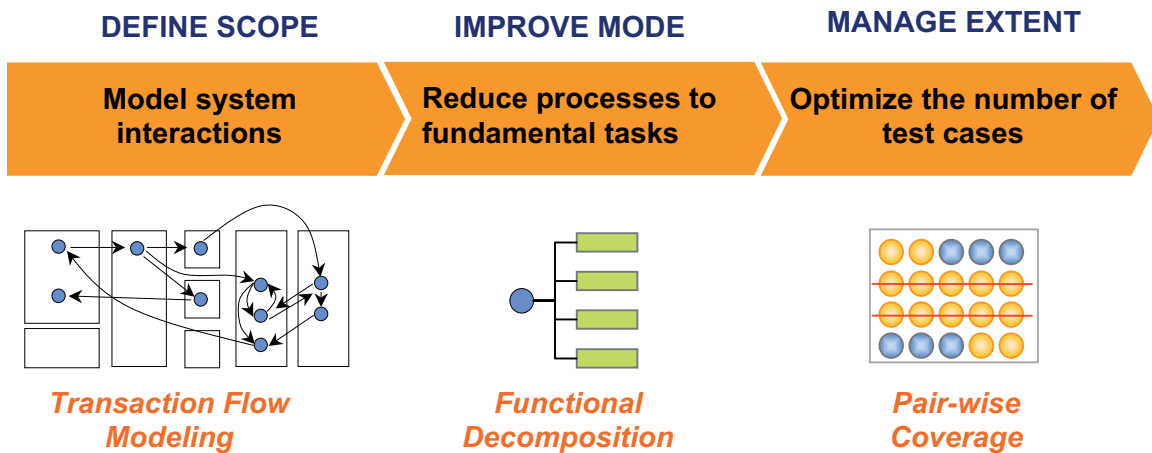
All Possible Combinations (8)				Pair-Wise Combinations (4)		
No	Side	Type	Time in Force	Side	Type	Time in Force
1	Buy	Market	Day	Buy	Market	Day
2	Buy	Market	Good till Cancel	Buy	Market	Good till Cancel
3	Buy	Limit	Day	Buy	Limit	Day
4	Buy	Limit	Good till Cancel	Buy	Limit	Good till Cancel
5	Sell	Market	Day	Sell	Market	Day
6	Sell	Market	Good till Cancel	Sell	Market	Good till Cancel
7	Sell	Limit	Day	Sell	Limit	Day
8	Sell	Limit	Good till Cancel	Sell	Limit	Good till Cancel

To apply pair wise coverage: if all of the pairs in a given combination exist in other combinations, that combination can be dropped. For e.g. we can drop combination no 8 as all the pairs (sell limit, limit-good to cancel and sell-good to cancel) occur in the 7<sup>th</sup>, 4<sup>th</sup> and 6<sup>th</sup> combinations respectively. By winnowing out the duplicates we can significantly reduce the number of test cases while ensuring that all pairs get tested in some combination or the other. Independent research shows that a system with 75 parameters with 10<sup>29</sup> combinations can be tested satisfactorily with just 28 pair-wise combinations. Our engagement experience has also shown that reduction in number of test cases through pair-wise coverage can significantly improve the speed and quality of testing.

### Putting it all together

Companies will have to take a fresh look at their functional automation process before making any further investments. Here is a structured approach on to how to put to use the techniques discussed so far.

## EXHIBIT 4



First, QA teams should begin by modeling the interactions between systems. Doing so helps break large functionalities to manageable parts and identifies all processing nodes and the flow of information and logic. The identification of all functional processing steps will form an input to the subsequent phases.

Next, all the functions and processing steps identified earlier need to be reduced to their most fundamental processes or transactions. Parameters describing these processes are then identified and test cases created. Logical groupings of test scenarios or test suites can also be formed by stringing together multiple test cases. This determines steps in the test process that can be automated with a worthwhile investment of time, money and resources.

Finally, using pair-wise coverage, the number of possible test cases can be pruned down. If it is found that some of the functionality or interaction flow paths will not be properly tested through this reduced set, additional test cases will then need to be created and added manually.

A test harness can be used to create and store the various processes, parameters and test cases in a reusable library. Such a tool can either have its own execution engine or interface with traditional tools, which a company would have already had invested in, for test case execution. Given the large number of processes and attributes, an algorithm might be necessary to bring down the number of possible combinations.

A recent client engagement demonstrates the benefits of such an approach. Overall testing and test result analysis effort has come down by more than 70 percent. Add to that a significant improvement in quality and quantity of test cases. By paring the effort it takes to create, manage and execute tests, this new approach now allows testers to focus on analyzing test results and leveraging them to refine or create new test cases.

While a fresh approach to functional testing will no doubt require upfront investments in analyzing the systems, detailing the processes and putting together a test harness, we believe such an investment will be worthwhile.

### About the Authors

[Eric Librea](#) and [Kanna Venkatasamy](#) are Principals with the Banking & Capital Markets Consulting and Solutions Group of Infosys; [Debashis Pradhan](#) is an Associate Consultant.

The authors wish to thank [Akash Gupta](#) (Senior Project Manager) and [Bharat Rao](#) (Programmer Analyst) for sharing their client experiences and contributing to the ideas presented in this paper.



For more information, contact [askus@infosys.com](mailto:askus@infosys.com)

#### About Infosys

Many of the world's most successful organizations rely on Infosys to deliver measurable business value. Infosys provides business consulting, technology, engineering and outsourcing services to help clients in over 30 countries build tomorrow's enterprise.

For more information about Infosys (NASDAQ:INFY), visit [www.infosys.com](http://www.infosys.com).