Implementing Automated Testing Solutions on Complex Brokerage Trading Platforms

Robert James Stanley
Team Lead/Senior Test Engineer
RTTS

The Brokerage Industry Challenge

Today, the brokerage industry is struggling to leverage the latest technologies to build complex order management systems, all while maintaining data integrity, security and timeliness to market.

Markets are interconnected in a river of money and electronics that span the globe. Financial companies are under mounting pressure to meet investor requirements for speed, accuracy and competitive pricing. Brokerage clients face a challenge in managing their global financial systems. Security, reliability and bandwidth on networks, software and infrastructure are increasingly under stress. As these networks and applications become more integrated and sophisticated, the need for quality assurance has never been greater. When defects go out into production the result is financial loss measured in time, money and expenditure of resources to correct the problem.

❖ A premier brokerage firm implemented a major overhaul of its trading systems. The firm updated legacy applications based on mainframe technology, to a state-of-the-art, client-server trading platform.

The firm had contracted with a software vendor specializing in such large-scale trading system conversions. As with many such projects, the implementation team had to contend with significant hurdles. The implementation plan had to accommodate:

- Maintenance of the integrity of the old trading system while simultaneously phasing in the new, ensuring uninterrupted service to clients and access to the marketplace.
- Verification that the new trading system complied with internal, Federal and State regulations governing the trading of securities.
- Validation that the new trading system properly handled and routed orders, maintained data integrity and accuracy, and provided real-time updates on orders and market conditions.
- Verification that the new system exceeded the old in key benchmarks such as speed of processing and updating orders, handling large volumes of market data, and presenting an intuitive interface for traders utilizing the system while also providing enhanced features to support different trading strategies.
- Incorporation of automated testing into the QA process for the new trading system.

The Strategy

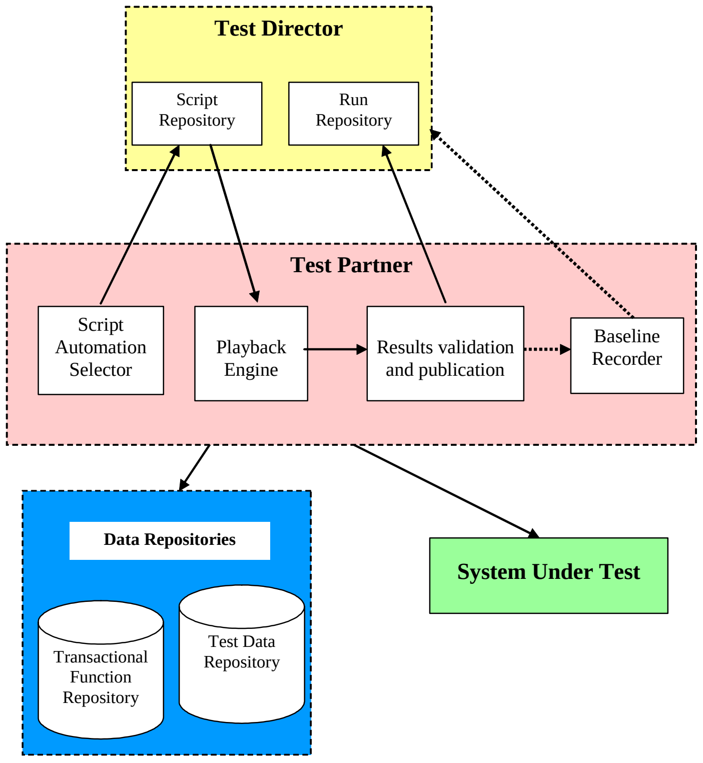
The brokerage firm knew it could not overcome these challenges without the use of automated testing but because of the technical complexity, the trading software had defeated previous attempts by automated test tools to create functional test cases using standard techniques. Eventually Compuware's TestPartner was identified as the ideal automated test tool to automate the trading application. TestPartner is a second generation automated test tool with technology to hook directly into the target application's COM (Component Object Model) components. This allowed for direct access to the application objects and controls that were either unreachable by other automated technologies or at best, only partially "seen" by operating with a third party interface. With this kind of open access, many of the major issues associated with automation were eliminated, including productivity costs due to poor object recognition and related automation code maintenance costs. Based on the performance of TestPartner, it became possible to plot a complete strategy to test the application and its systems.

- Spin the Server

The trading software communicated between clients and the server on a transactional basis, with a request/reply communication model between the client and server. By being able to take advantage of TestPartner's ability to use the trading application's native COM controls, TestPartner could "talk" directly to the trading server, a feat not readily accomplished with other automated test tools. Thus, test scripts could be created to spin orders directly into the server (bypassing the application GUI) and move them through the order management system with client/server transactional messages and queries to both to the system's real-time database as well as SQL to the backend booking systems. This approach was selected because the server architecture was relatively stable while the GUI design underwent regular revision. The result of bypassing the GUI, but using the client protocols for communicating with the server, was that any patches or builds from the software vendor that dealt with the backend systems could be tested and validated quickly, without the maintenance burden of GUI-based automation code.

- GUI Interrogation

One of the most common reasons for the failure of test automation implementation projects is that naively built test scripts lead to an unanticipated and unacceptable code maintenance burden. The typical strategy is to drive test cases through the GUI by clicking buttons, entering text into fields, selecting menu options, etc. However, testers untrained to leverage the power of testing tools are forever playing catch-up with the GUI, as features change regularly due to user requests and development enhancements, thus breaking the automation code for playback until it can be retooled to be compatible to the most current release.



The trading software on this project had additional factors that foiled previous attempts at automation. The first was its very complexity with numerous controls and data feeds. The second was that it could be counted on to have potentially dozens of releases and upgrades from the software vendor on a yearly basis, after which it would undergo customization to fit the requirements of each brokerage client and locality. This meant that the application could be counted on to remain dynamic, especially in regards to the interface.

In this area, TestPartner had an advantage over competing tools because it didn't have to guess what type of object a GUI control might be. Because TestPartner could "see" directly into the ActiveX controls, automation engineers could always script against control properties and methods accordingly. Functions could be written in TestPartner to take advantage of the COM object properties and methods, to create a highly modularized automation approach. If the object changed in a release, then only a function in a TestPartner code library needed to be updated and significant degree of code reusability was achieved.

The combined strategy for GUI verification was to drive transactions through the server and validate resulting code changes by interrogating the GUI front end for the transaction data submitted to the system. Since the bulk of activity that affects the GUI can be driven and controlled though the server, test engineers could drive orders through the trading system and easily monitor the front end for a change in status to those orders. Order entry, amendments, routing, trade executions, busting of trades, the entire spectrum of brokerage functionality could be run through the more stable server environment via the backend pipeline, and verified in the front end in the data grids that displayed order information. This helped insulate and 'future-proof' the automated test scripts against changes since server processes underwent far fewer upgrades and builds than the GUI itself.

- GUI Inventory

In addition to the work described above, a more general approach to GUI testing was not abandoned. Instead, a complementary strategy was devised where the 'look and feel' of the GUI was recorded on the application level, in which the UI properties and features of individual objects and controls were interrogated by automation code, rather than through the use of button clicks via a mouse or keyboard. By tapping into built-in API's and libraries native to the software, automation engineers were able to establish a baseline of all the application objects and controls. This avoided the maintenance headache associated with automation scripts replete with mouse coordinates, synchronization issues and the other usual pitfalls associated with automated testing. If objects, controls and properties changed in the application, then a tester would simply run a test script designed to inventory everything in the application and update the baseline automatically. The decision was made to keep user-level testing of the GUI on an as-needed, manual basis, because the ROI and the maintenance overhead for automated test scripts for that type of functionality was not cost-effective and did not add enough value. Because of this approach, a level of testing was achieved that would not have been otherwise possible. In the automated GUI testing that was implemented, TestPartner was able to drill down to as fine a detail as the brokerage client deemed necessary to test the standard look and feel of the application (including font size and color used in its GUI controls) thus ensuring a stable, successful process from release to release.

Results analysis

The results of automating the trading software proved a success by the brokerage's own metrics. Playback of the automated regression test bed completed in 8 hours what had previously taken one manual execution resource 8 days to run. The brokerage client calculated that by simply executing the automated suite of tests in only four cycles of testing, it paid for itself. In addition, the client was able to reassign their business analysts, who were being used as manual testers as a stop-gap measure, back to their regular duties, an additional cost savings in time and resources.

As an additional bonus, the engineers were able to design and implement a database for input data and fully integrated it to work with the TestPartner suite. This feature allowed for a more robust solution, with better support for future expansion and maintainability than typical comma-delimited files or spreadsheet programs.

Finally the engineers, at the request of the client, integrated TestPartner to work with Mercury's Test Director, the firm's defect-tracking and test management software. TestPartner was able to take full advantage of Mercury's published API for Test Director and a robust integration between the two products was built. TestPartner is able to login to Test Director, read in test cases, update results and set pass/fails for test steps and entire test cases all via Test Director's API.

Because of TestPartner's ability to work with Test Director, a final design feature was created to put the power of automation into the hands of non-technical users. By embedding a keyword layer in the TestPartner code, the firm's team of testers could create test cases on-the-fly and kick off playback from TestDirector. Manual testers could create test cases in Test Director as they normally would, using phrases that, when read in by TestPartner, are translated into a direct action to be performed by automation. Business-friendly keywords were layered into the steps that make up the test cases stored in Test Director. The non-technical tester identifies the test cases

built this way to run in Test Director and kicks off playback in TestPartner. TestPartner retrieves the test cases that the user has specified, interprets the steps the user wants to perform and then executes the specified transactions. Later, when validation is being performed, TestPartner goes back into Test Director, retrieves the expected result and compares it with the actual result, and publishes the information back into Test Director, updating the pass/fail status of the test.

TestPartner was able to provide a total solution to what had been a testing quandary. By providing seamless integration with both the application under test, and with the defect tracking software, TestPartner was able to not only tie the two more closely together but actually led to a unique testing solution. It leveraged existing client assets (for which a substantial investment had been made) and actually expanded on the usability and out-of-the-box capabilities.

The success of this project has encouraged the client to pursue automation in other areas previously thought inaccessible to automated test tools. As is often then case, coming up with the right strategy, developing a viable test plan and having the right expertise on hand were the key factors in overcoming the difficulties in automating complex trading platforms.

About the Author

Robert Stanley is a Team Lead and Senior Test Engineer for RTTS. With 13 years of management experience and 6 years experience leading successful RTTS testing engagements, Robert has worked with many Fortune 500 firms in the brokerage, insurance, pharmaceutical, reinsurance and software development vertical markets.

About RTTS

RTTS is a professional services organization that specializes in the testing of IT applications and architecture. With offices in New York, Orlando and Phoenix, RTTS has been serving Fortune 500 and mid-sized companies throughout North America.   Drawing on its expertise utilizing the leading testing products, expert test engineers, and a proven methodology to provide the foremost end-to-end solution, RTTS ensures application functionality, reliability, scalability and network performance. For more about RTTS, please visit our web site at www.rttsweb.com.