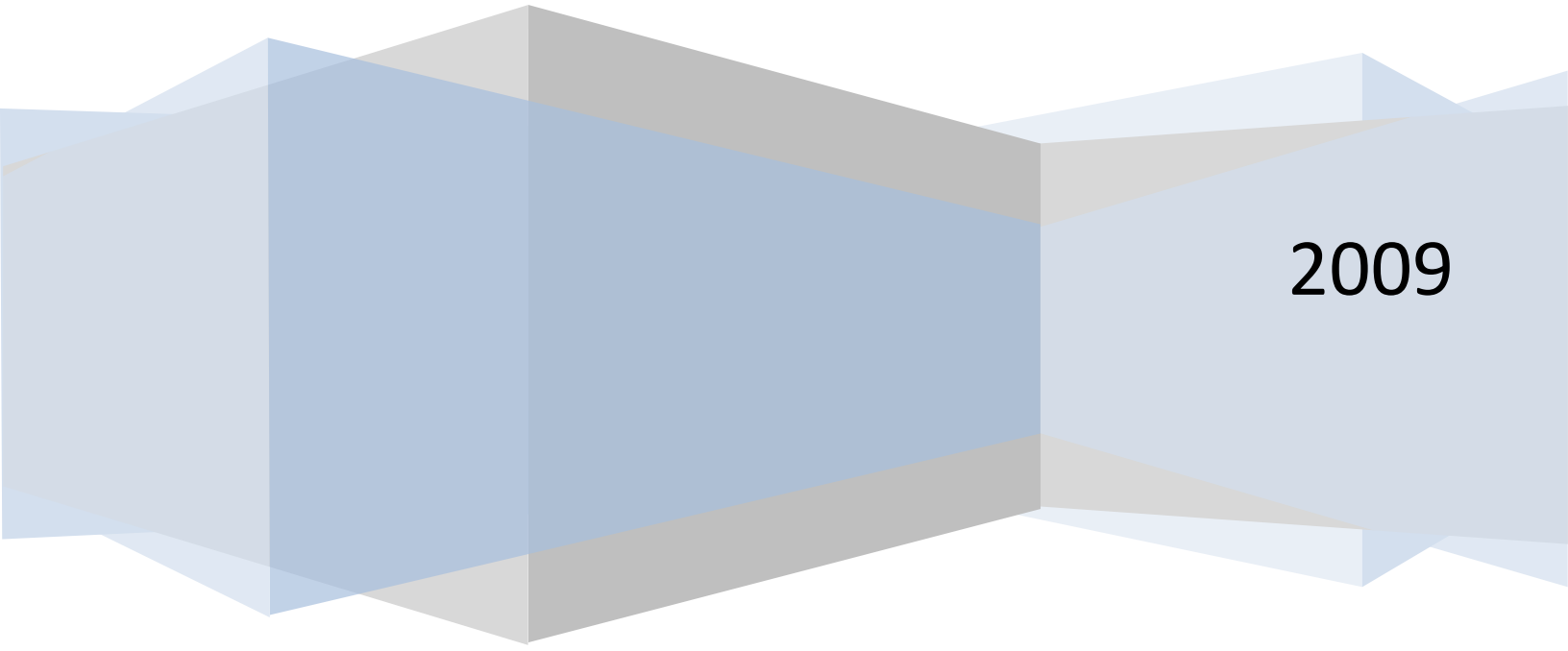
Advanced SOR Testing

Addressing the challenges in testing advanced execution platforms

By Iosif Itkin, Senior NFT Analyst; Alexey Zverev, Senior R&D Manager; Olga Buyanova, QA Analyst; Dasha Klyachko, Managing Director



2009

Table of Contents

INTRODUCTION

Regulatory changes, fast technology improvement and the appearance of new execution venues have added more complexity to the European equity markets. The challenges of ever increased liquidity fragmentation in a highly competitive industry dictate that companies commit to significant investments into advanced execution platforms. The absence of strong Quality Assurance can delay the development and deployment of enhanced trading systems and/or have other negative impact, such as missed defects.

In the introduction part of this document, we discuss why special attention to testing of the execution frameworks and algorithms incorporated into trading software is so important.

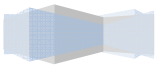
The first generation of Smart Order Routers (SOR) is being replaced by solutions containing sophisticated customizable algorithms. The correctness of their work directly impacts profitability and compliance with various regulatory requirements. It is important to emphasize that such solutions cannot remain stable for long periods of time in the fast changing markets. Algorithms used in SOR need constant adaptation to reflect the latest changes. It is not uncommon to see systems with release cycles shorter than two weeks, as fierce competition dictates the necessity to shorten time-to-market. This goal cannot be achieved without efficient quality assurance procedures in place.

For any trading system these days, high throughput and low latency are no longer “nice-to-have” features, but rather crucial requirements. It is especially true for advanced SOR systems, since their very purpose is to catch available liquidity. Without accurate latency measurements one cannot be confident in a system’s ability to allow its user to stay ahead of decisions made by competition. The main throughput challenge for modern execution platforms comes from a dramatic increase of reference market data volumes. The systems must process this data as fast as possible to have a relevant snapshot prior to making routing decisions. Load and stress testing must be carried out to verify that SOR will be able to withstand growing volume spikes.

Complex execution platforms can be distributed across multiple application/server instances and datacenters. Quite often different versions of the algorithms/software are simultaneously deployed on different instances of the application. To satisfy requirements, advanced SOR needs to provide load balancing, fault tolerance/high availability, and resilience to unexpected events. Monitoring and deployment procedures should be tuned in such a way as to guarantee the fastest possible response. In summary, a large, complex trading system means significant operational challenges. By conducting Operational and Non-Functional Testing (NFT) in a test environment, QA can red-flag possible production issues in advance and thus help the Support Team be prepared to resolve them quickly and efficiently.

Although comprehensive testing of the SOR systems leads to obvious benefits, it is way too often omitted by the stakeholders. The main reason for that is that testing of such systems is a non-trivial challenge.

The first part of this paper discusses the most common issues encountered as the testing of advanced SOR testing is being considered by the stakeholders. These quality assurance challenges can be effectively addressed when the following three key elements are in place:



- Advanced test tools;
- Advanced testing strategies;

- Advanced engineering staff to apply both.

The second part of this document looks at the requirements for such test tools and testing approaches. As the authors specializes in testing and development of complex financial applications, we have included a short description of the relevant tools, project management processes and staff skills in the form of a Case study, which shows a possible approach to achieving quality objectives for advanced execution platforms.

1. SOR TESTING CHALLENGES

1.1 Client order flow

In order to test a Smart Order Router, one needs to submit client orders into it. Even when the normalization of input flow is in place, one still needs to cover all systems responsible for its generation: web and thick clients, direct API, and Order Managing Systems.

Test scenarios must cover all asset classes and security types; all order sides, types, TIFs, and handling instructions. Additionally, not just new orders, but also change-replace (amends) and order cancellations must be tested. The latter two are applied for orders in various states, e.g. new, filled, partially filled, TLTC (too late to cancel). The incoming order flow should comply with a certain instrument, price and size restrictions.

In addition to positive use cases, test scenarios should also account for erroneous incoming messages as negative test scenarios.

1.2 Static Data

Testing of trading systems heavily relies on static data: instruments, participants and prices. Ticker symbols and market data subjects vary for different exchanges. The set of accessible instrument changes from time to time. End-Of-Day (EOD) prices are updated daily. To be able to execute automated tests, one needs to include appropriate symbol names into the test data. Prices used in the test scenarios should be within appropriate limits. Therefore, one needs to constantly update the test data and be cognizant of the fact that the data itself is rather complex.

1.3 Exchange Simulation

To test a SOR system, one needs to place it between the client order flow and the exchange simulator. Exchange simulation is required to provide responses to child orders and amends and cancel instructions generated by SOR. Exchange simulators should support protocols used by the SOR to access execution venues. In addition to providing consistent response to child orders, the simulators need to send self-consistent market data that corresponds to changes in the transactional flow.

1.4 Diversity of Markets

The test framework needs to take into account the differences between various exchanges, dark pools and other execution venues. Each of them have their own trading times, different algorithms for open/close/unscheduled auctions, crossing rules, and order types (market/limit, various types of pegged orders, available TIFs).

Concurrently acting simulators should introduce different latency before response to replicate the difference between the real systems.

The exchanges use different symbol names, and trade in different currencies. Various tick, unit and minimal order sizes are valid depending on the exchange.

1.5 Impact of Reference Markets

The markets are no longer independent from each other. Observations show that a strong correlation exists between them. MTF often uses information from the home or aggregated pan-European (North-American) market to limit price range available for execution or the execution price determination. Exchange simulation should take this impact into account.

1.6 FX Feeds Simulation

Due to the fact that trading ultimately involves multiple currencies, the test environment should provide the means for accurate FX Feeds simulation. It is important to keep stock prices in various currencies consistent with the current FX rate. The latter should not be static for realistic testing.

1.7 The Universe of Testing Scenarios

The ultimate challenge of SOR system testing lies in the limitless possible responses by the exchanges to child orders. An order can catch available liquidity, or the liquidity might go away prior to when the order reaches the order book; additional liquidity can become available when the order reaches the order book, and the state of the current and reference markets can change after the order has been executed. If one multiplies these options by the multitude of re-routing options that are possible in an advanced SOR, it becomes apparent that it is necessary to consider a virtually countless number of test cases.

1.8 Quest for Hidden Liquidity

SOR systems perform liquidity pinging / scanning for hidden liquidity (i.e. they attempt to interact with invisible/non-display, iceberg/reserve, and discretionary orders). One needs to carefully simulate a realistic response from the venues to these strategies. In order to do it, QA engineers need to possess an in-depth understanding of the dark liquidity concepts.

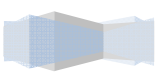
1.9 Reliance on the Adaptive Database

Some routing algorithms for venues priority are based on historical databases (historical statistics on trading patterns per symbol and per venue). Some routing algorithms for venues priority are based on the real-time adaptive database. There are also combinations of the two: some routing algorithms for market centers priority are based on historical probability patterns and real-time adaptive opportunity patterns.

Reasonable tests scenarios should pre-fill historical databases with relevant data and provide realistic dynamic flows to initiate real-time processing.

1.10 Stochastic Behavior

It is not uncommon to see algorithms containing a certain degree of randomness in them. From the testing perspective it means that the system might have more than one correct predictable behavior for a given list of inputs. Therefore, pass/fail criteria become much more complicated. Sometimes stochastic behavior is explained by concurrency effects and the time dimension impact. That is why it may manifest even when no random number generator is used.



1.11 End-Points Synchronization

The system might not have a single focus end-point, where all inputs or outputs are concentrated and processed. It may be necessary to simulate several input protocols, while protocol implementation details

might be undocumented or hidden. The simulation might also require compatibility with different software platforms.

It is necessary to run all interactions with the system under test from a single point to ensure that all data fed into the system is consistent and synchronized. Incoming order flow initiation should be performed in sync with decision start. The task can be even more complex if the static inputs are unacceptable for the models used in advanced SOR. In that case, input flows should be coordinated with the start of evaluation cycles. Particular branches of model code might work only when particular inputs come in a pre-defined order. Events close to each other timing-wise can change time-order in a multi-processor and especially in a distributed environment. The test tool should provide the means to simulate the required behavior and ensure that a particular branch is working.

1.12 Precision of Latency Measurements

It is necessary to take into account all components affecting delays within the SOR system, in particular networks and firewalls, transport protocols and application level latency. Apart from challenges common to most trading platforms, additional problems could be pointed out that are caused by the complex behavior of algorithms and difficulties related to tracking the relationships between parent and child orders. This can be done by using information obtained from system logs or network sniffing.

1.13 Real-Time Data Volumes

The capacity of the test tools should not be inferior to the capacity of the system under test. In particular, one needs to simulate tens of thousands updates per second and still be able to analyze system's behavior. Not only real-time performance is of importance, but also the analysis of regulatory reporting generated by SOR, such as SEC Rule 606 (former Rule 11Ac1-6) Disclosure of Order Execution and Routing Practices for Non-Directed Orders, OATS, OTS, MiFID, etc. This means that the test tools should be capable of processing of a considerable amount of information that might be hosted on less scalable hardware than the SOR itself.

1.14 Environment Maintenance

Advanced SOR is usually a highly customizable solution deployed across multiple servers and connected to various feeds and databases. Deployment, upgrading, monitoring and configuration management of the test environment may require significant effort by the Quality Assurance team. The test system in some respects can be even more complex than production, as it includes client order generation and exchange simulation facilities. Testing might also require changes in static data and adaptive databases that differ from those that take place in production.

2. ADVANCED TESTING TOOLS

2.1 Test Harness Requirements

Automated testing of a distributed trading backend present in most SORs and advanced execution platforms can only be performed by an enhanced Test Harness solution. Such a testing tool must allow interaction with the components inside the system via different network connections and APIs.

In this section, we outline the requirements for executing end-to-end functional and non-functional tests.

The Test Harness needs to possess the trading framework that allows simulating inbound orders/cancellations and amending the flow via multiple connections, outbound executions flow, market data recording and replay, clearing and settlement connectivity.

The Test Harness needs to incorporate a complex events processing engine and support multiple trading (FIX, FAST, Fidessa, SSL, etc.) and network (TCP/IP, UDP, Multicast, HTTP(S), SOA, Tibco RV/EMS, Message Queues) protocols out of the box. It should be expandable in order to satisfy requirements to cover more protocols.

In addition to client and server simulation, the tool needs to provide powerful monitoring and the ability to drill down to particular captured messages.

The frameworks within the Test Harness need to support logs processing and standard application servers monitoring.

Remote environment management can reduce the load on quality assurance staff during test execution.

The engineers need the ability to write script test scenarios using a commonly distributed programming language: VB, Java, C#, Perl, TCL, Python, etc.

Extensive test library should support the following:

- Managing connections to the system;
- Running multithreaded algorithms controlling different event flows between the Test Harness and system under tests;
- Logging events and measurements and collecting statistics.

The Test Harness also needs to have a flexible reporting subsystem allowing report generation in MS Excel or other format.

Our experience shows that the following features are extremely useful for functional testing of the SOR systems:

- The ability to develop test scenarios consisting of test cases;
- The ability to record real environment behaviour and create test actions reproducing observed events;
- The ability to articulate test cases as sets of actions described in Excel spreadsheets (such scripts can be reviewed and maintained by business users and analysts);

- The ability to control the entire test environment from a single script;
- Integration with one of UI testing tools.

For performance testing, the following features in a Test Harness are essential:

- The ability to emulate the load caused by a number of different virtual algorithms hitting the system simultaneously;
- The ability to pump pre-recorded message flow into several network connections (either client or server);
- Server simulation capable of processing high volumes of incoming requests;
- A reporting subsystem allowing for the analysis of network capture/server logs and generating consistent and clear reports within a reasonable amount of time.

2.2 Exchange Simulator

Exchange simulator is a tool whose behavior approximates that of a real exchange to the maximum degree. It accepts all types of requests supported by a specified exchange and acts as an order book according to corresponding rules. The Simulator completely supports all trading phases applicable for a market, such as opening/closing auctions, etc.

The simulator accepts order requests and provides exchange responses via a standard order management link, such as FIX. It also publishes the order book state and other parameters via the Market Data feed.

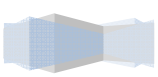
As a testing solution, the simulator also provides the ability to control the market simulated during a test.

The following are the features for controlling the simulated order book:

- Historical prices. This standard feature is used to replay historical prices by generating artificial events and populating order book according to previously recorded historical or artificial market data.
- Managing market impact. All external orders are going to the same order book and therefore moving it away from its historical state. An external market driver constantly monitors the order book; if the book significantly differs from its historical state, the driver generates a counter flow of artificial requests in order to compensate the external impact and move the market back to its historical state. The market driver can also be used to simulate a complex market behavior or a real market impact. The latter is useful for behavioral testing of complex trading algorithms.
- Simulating predefined market conditions. The market state is controllable for the external applications, such as test scripts or test harnesses. The simulator accepts requests to setup the order book to a specified state. This is needed for building a controlled Exchange Simulation environment for functional tests, which verify the system's behavior under specific market conditions.

A very important requirement for a Simulator used in the testing of a SOR system in a fragmented market is the ability to concurrently simulate many different instruments across different exchanges. It is crucial to have a synchronized simulation of multiple different markets and have the ability to control them from a single test scenario. In addition, different markets may introduce different latencies for requests and

response delivery. Therefore, the Simulator must provide the ability to introduce different delays for different markets.



3. ADVANCED TESTING STRATEGIES

3.1 Regression Testing

By regression testing we understand executing test scenarios that cover all system functionality in its entirety. Regression testing is done to verify that no issues were introduced during the implementation of a new version of the application or during the customization of the system. Regression testing is a standard technique used to guarantee quality of every complex system. Thoroughness and skill can turn this approach into an advanced testing strategy. The development of test cases for regression testing requires an in-depth understanding of the system and the business processes. We recommend that regression testing start by executing scenarios related to the following:

- Bug fixes done in the previous release;
- Areas that frequently contained defects in the past;
- Areas which have had recent code changes;
- Areas that can significantly affect algorithm profitability.

Sufficient code coverage during a regression run can be achieved when the test team:

- Has access to protocol specifications;
- Understands trading system architecture and market model;
- Is familiar with the new/old system requirements and test scripts history;
- Constantly monitors the bug tracking database paying particular attention to defects observed in production.

A typical SOR regression test suite covers the following areas:

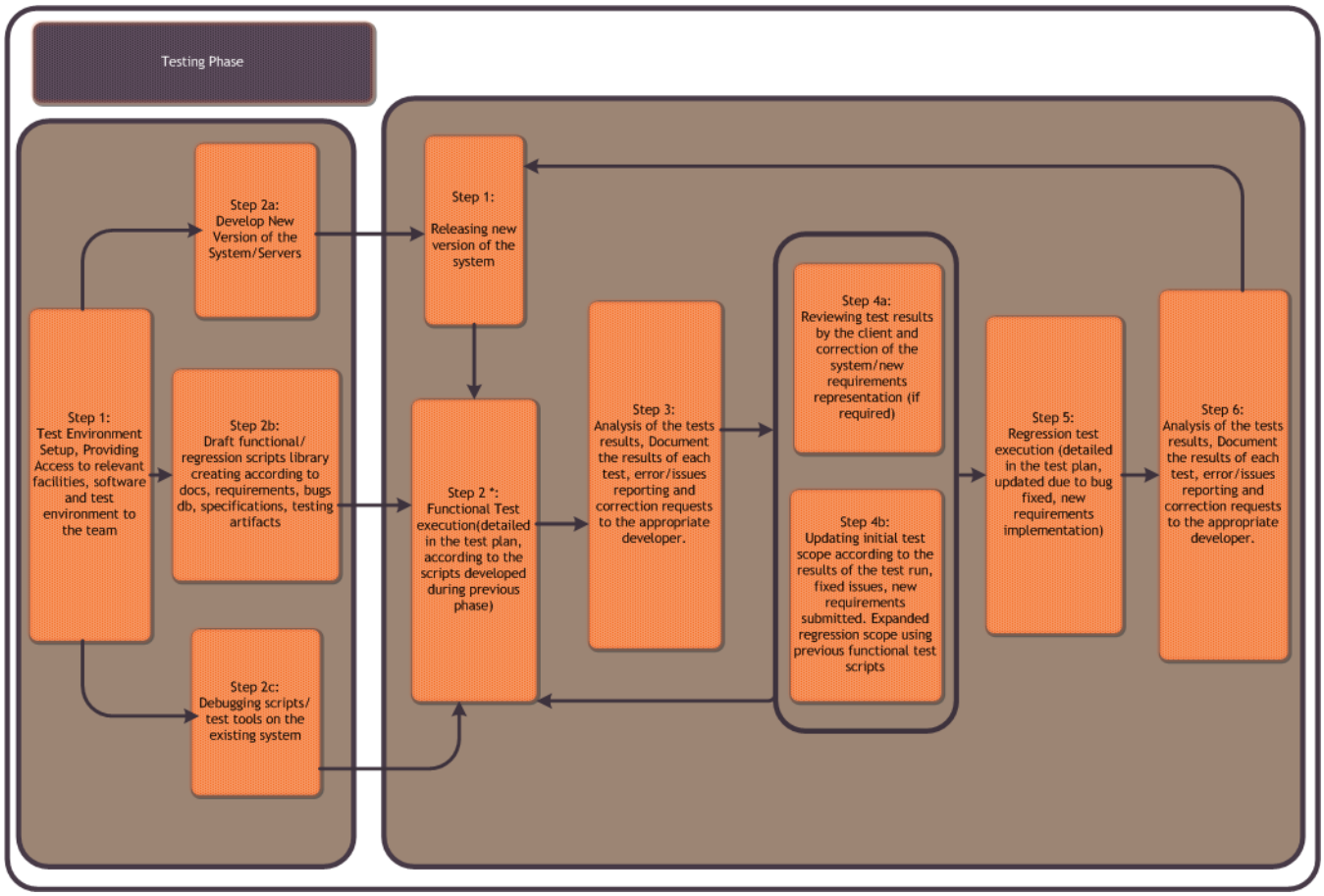
- All applications sending orders to Smart Order Router (e.g., web client, thick client, API, etc.);
- All SOR modes supported (e.g., fire and forget, continuous re-route, etc.);
- All asset classes and security types;
- All order sides, types, TIFs, handling instructions;
- A certain range of typical order sizes and prices (marketable and non-marketable orders);
- All trading venues, including all public market centers protected under Reg NMS and all dark pools available (also consider requirements for internalization);
- All trading sessions and daily activity patterns for all venues (including trading halts, etc.);
- All business rules and routing algorithms, including but not limited to:
    - Routing algorithms for venues priority based on historical database (historical statistics on trading patterns per symbol and per venue);
    - Routing algorithms for venues priority based on real-time adaptive database;
    - Combination of A and B: Routing algorithms for market centers priority based on historical probability patterns and real-time adaptive opportunity patterns;
    - Routing behaviors for venues priority based on explicit customer preferences that override default system settings (customized algorithms such as exclusion of certain venues from routing choices, limiting max size of child orders, etc.);

- Liquidity pinging / scanning for hidden liquidity (attempting to interact with invisible/non-display, iceberg/reserve, discretionary orders)

- New orders, change-cancels (amends), cancels;
- Full fills, partial fills followed by rejects or cancels or posting of the balance of the order on the venue book, TLTC (Too Late To Cancel) scenarios;
- Order validation and error handling specific to Smart Order Routing;
- Failover and Recovery scenarios, such as handling of disconnects from specific venues or market data feeds (also consider scenarios for triggering self help rule against a certain market center when detecting repeated delays in response to incoming orders);
- Regulatory reporting of order traffic generated by Smart Order Router (such as SEC Rule 606 (former Rule 11Ac1-6) Disclosure of Order Execution and Routing Practices for Non-Directed Orders, OATS, OTS, MiFID);
- Verification of the log files, trade and quote databases to facilitate a comprehensive audit trail and "compliance snapshots" (the capture of contemporaneous trading and quote information to substantiate order routing decisions).

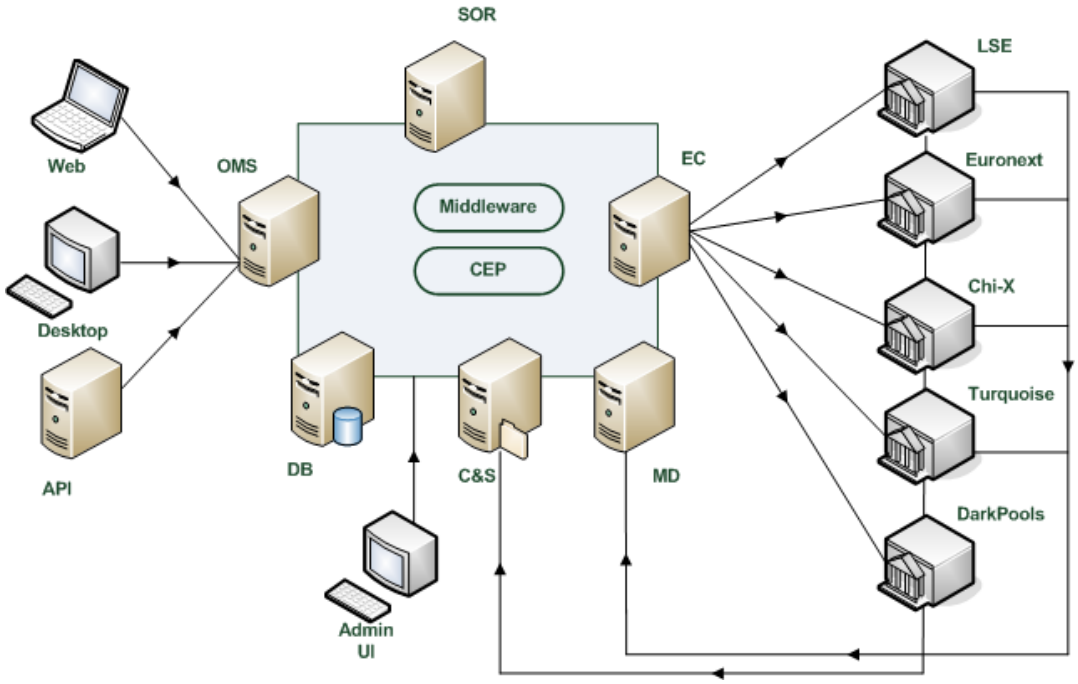
It should be noted that the applicability of individual items from the above list is directly related to the functionality supported by the system under test.

The diagram below shows a generic project flow and the relationship between functional and regression test execution:



3.2 Back End Testing

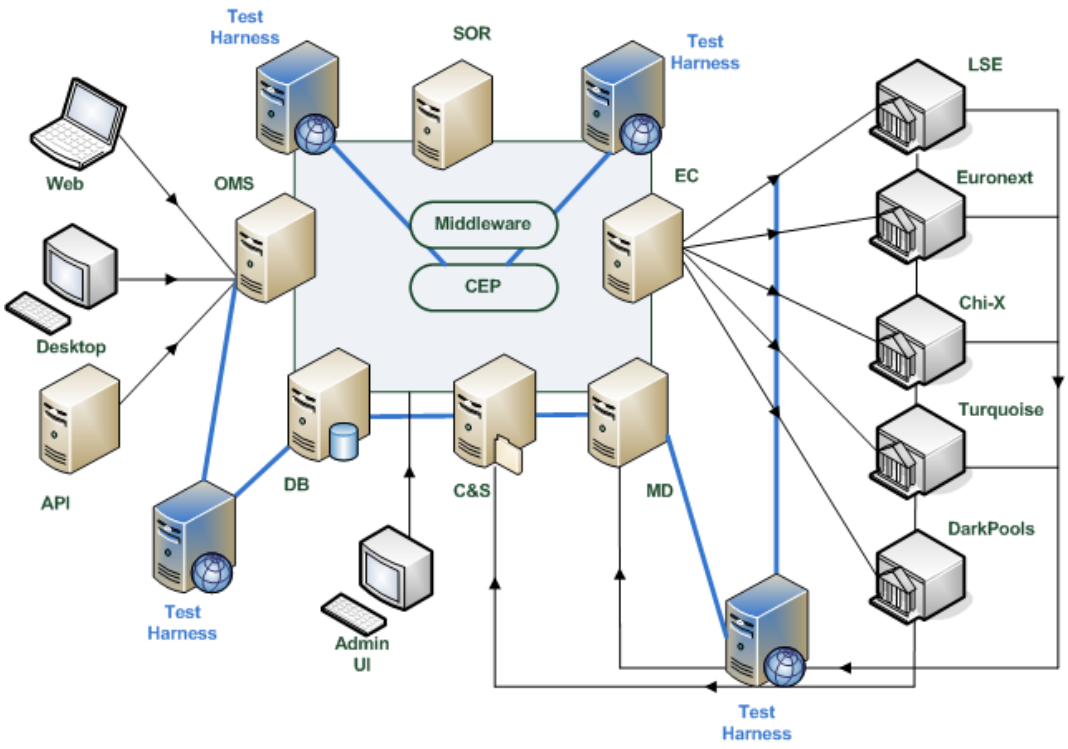
The diagram below shows a simplified schema of a trading system. We will use it to illustrate concepts of backend testing.



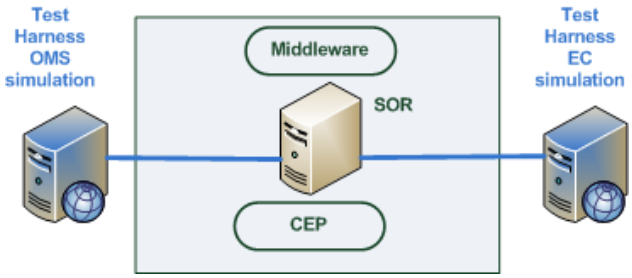
In the diagram, we have an Order Management System (OMS) responsible for obtaining orders from Web, Desktop and API clients. Orders are passed to the Smart Order Router (SOR). Child orders generated by SOR go into the EC (Exchange Connectivity) layer. The system receives MD (Market Data) and C&S (Clearing & Settlement and Static) data. DB (Database) is used for persistence and regulatory reporting. Admin UI is used by the system's operator to perform various maintenance tasks.

The most common type of testing relies on using available GUIs to communicate with the system. Often UI Automation tools like HP QTP, Borland SilkTest, Selenium, etc. are used to emulate end user activities. Although this kind of testing is crucial for any trading system, it has several limitations. The main difficulties are related to maintaining the state of the system under test and exchange responses. With GUI testing it is sometimes difficult to pinpoint a component causing a problem or test different servers in isolation. There are also scalability limitations on the number of scenarios that can be executed from a single workstation within a given period of time.

Backend testing addresses these issues by "drilling down" into the system's backend and communicating with servers on the protocol level.



Test Harness used for backend testing allows for monitoring communications within the system without replacing an actual component. Alternatively, it can be used to emulate client order flow and execution venues. The support of internal protocols allows performing unit testing. The diagram below shows one of the components (in this case SOR as a whole, but in reality it can be a part of the SOR) surrounded by Test Harness simulating low level communication:



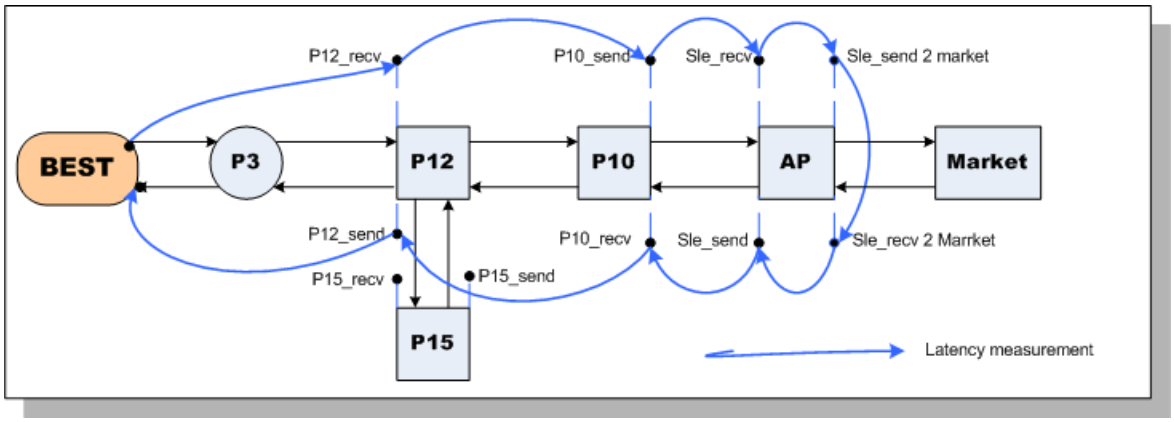
3.3 Performance and Latency Measurements

Obtaining accurate figures for system's capacity and latency is a complex task that needs to be carefully planned, prepared and executed. We are not going to cover all aspects of performance testing in this document, but will rather concentrate on several best practices that benefit SOR testing. These are:

- Analyze system requirements and carry out static analysis;
- Outline load business model;
- Collect monitoring information from several sources;
- Invest into monitoring and test results processing;

- Use scalable tools for order flow generation and market simulation;
- Ensure that an iterative process is in place.

System requirements should be identified to drive performance testing and optimization processes. Such requirements need to be critically analyzed to ensure that they are clear, verifiable, realistic, and that they have business meaning. As they are being captured, the requirements should be separated into those that are related to spikes and those that are related to sustainable processing. This will help avoid expending efforts on trying to reach unobtainable targets for systems capacity. It is important to emphasize that, for SOR systems, latency figures during the spikes are extremely important, as the spikes represent market movement. For latency measurements, it is recommended that requirements for overall processing and internal delays are identified, and that the focus is on paths that present the main value for business. For example: in many cases forward order processing delays are much more critical compared to propagation of information related to acknowledgements and fills. The diagram below shows various delays measured for a GL Trade-based system:

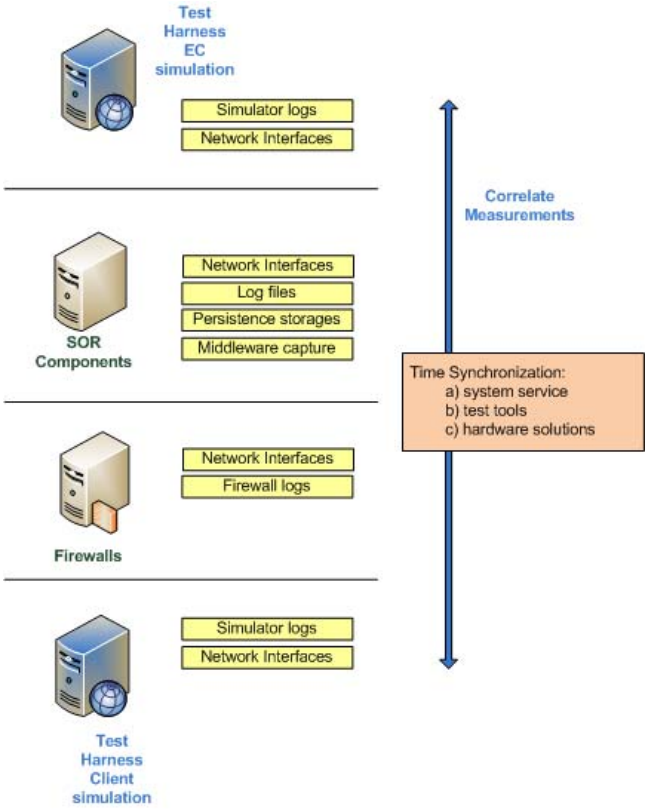


While identifying SOR system requirements, one needs to think in terms of parent and child order instructions, reference market data intensity, and response per order rate. Child order flow is the most complex, as order flow generation tools control only parent orders, while child orders generation heavily depends on the SOR logic.

Static analysis of requirements allows predicting some effects even prior to launching a test, e.g. lack of disk space for log files will definitely result in system's unavailability over a period of time. We highly recommend using static requirements analysis while planning performance testing.

When executing testing for advanced execution platform, one should not concentrate on using only simplistic scenarios based on sending/amending identical orders. Instead, we recommend identifying the underlying business model that describes types of participants using the system and their behavior. Some SOR algorithms may be used more frequently than other, various sizes and constraints might be passed to various algorithms. Inbound orders flow generator should reflect the heterogeneity of the real world. Underlying business models simplify the process of communication of the actual executed test scenarios to the appropriate stakeholder. Ideally, the load model should be customizable, so that its output can be tuned without modifying the model itself. The comparison of the model output against production log analysis provides a valuable insight regarding model characteristics.

The information related to messages that pass through the system can be obtained from various sources. The diagram on the right shows the main locations of such information: Network interfaces, log files, middleware capture, persistence files and databases. Monitoring tools need to extract lines/packets related to a single message or business transaction from various parts of the system, group them and compute the aggregated data. We believe that several sources need to be used even if all of them provide similar information. For example, the same information can be present in persist files and network capture, but the reconciliation of the sources can provide extra hints regarding the time required to support persistence and can also decrease the probability of results processing errors.



The effort invested in system monitoring can pay off not only in test environments. Customizable monitoring and results processing can also help analyze production information (although simulator logs will not be available in production). Usually, we generate two types of reports. One of them is automated; it is generated based on the results of a particular test run. The other type aggregates information from several test runs; the results differ either in terms of the load level or the configuration of the test environment.

While looking into the roots of the bottlenecks reported by the test tools, one would like to avoid issues caused by the testing tools themselves as opposed issues caused by the deficiencies of the system under test. Scalable test tools allow generating required volumes by using available hardware. There have been instances when we had to use pre-recorded data for incoming message flow generation and simplified crossing algorithms in exchange simulation. Very often, the most time-consuming step in performance testing is the processing of test results. Therefore, a fast processing tool can be beneficial for the entire project.

We are confident that performance testing and optimization should be an iterative process. It should start at the earliest possible stage, even prior to implementation of a complete business model. Requirements identified at an early stage of a project should be verified again against the actual test results. It is also recommended that the testing team always plan for several optimization cycles.

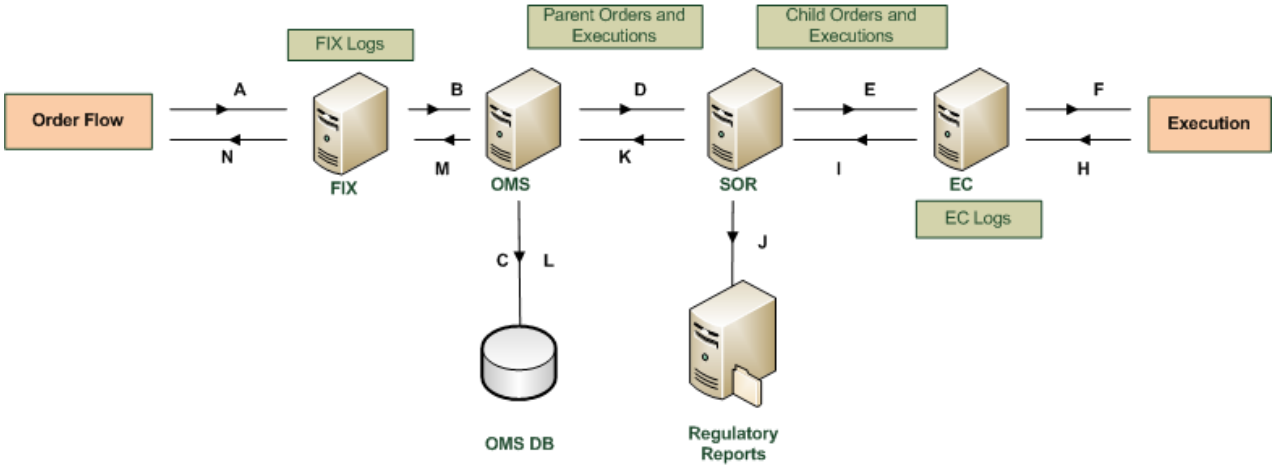
3.4 Reconciliation Testing

As there is a limitless set of possibilities for the paths within advanced SOR, we need to consider approaches different from those used by scenario-based regression testing. One of the testing approaches that has proven cost efficient is reconciliation testing.

A standard test scenario consists of steps and expected results. In case of a SOR system, test steps will include:

- Pre-configuring the initial state of SOR;
- Submitting predefined client orders;
- Simulating predefined exchange responses;
- Monitoring the behaviour of the SOR system and verifying orders that it generates/re-routes and responses to the client.

Reconciliation tests make no assumptions on inbound client order flow or exchange responses. Instead, they verify data consistency between various points in the system using a set of customizable rules.



Let's consider a few simple examples:

- The quantity of executed child orders should match the quantity of executed parent orders (**I** vs. **K**)
- Each order and execution must to be stored into the OMS database (**A** vs. **B** vs. **C**)
- Each execution should be reflected in regulatory reports (**I** vs. **J**)

All of these rules are very straightforward and easy to check. However, much more complicated scenarios can be coded. A good example would be verifying aggregate characteristics of generated child orders vs. the parent order. The complexity and variety of the checks (**D** vs. **E**) reflects the complexity of advanced SOR execution decisions.

We recommend using software developers to implement a reconciliation testing procedure. Still, the output of the verification should be presented in pass/failed criteria of a test case (accompanied by a detailed discrepancy report). Boolean test case output should be stored into the same results repository as the one that is used for ordinary automated tests.

One of the additional benefits of reconciliation testing lies in the fact that the same procedures can be used in production environment to verify data consistency between components.

3.5 Behavioural Testing

In this section we address one of the approaches to testing stochastic systems. Behavioural testing concentrates on aggregated metrics that reflect the behaviour of a system. We put the advanced router into various dynamic conditions, execute particular test sequences and calculate the targeted metrics. The test should be repeated several times to verify the stability of the metrics for the same inputs and the elasticity towards small fluctuations in inputs:

- Replay historical or artificial market data;
- Make Exchange Simulators respond according to market conditions being replayed;
- Generate a client order or a set of client orders coming into the SOR module;
- Analyze the behaviour of the SOR module and assess the algorithm performance in terms of the average price, latency, fill probability, total commission and other relevant metrics for a single order or a set of orders.

Successful behavioural testing requires additional sophistication from simulation and inbound order generation procedures. The execution of child orders should be strongly correlated with the market data that flows into the system, and market impact and unpredictable market movement simulation are also essential.

Metrics calculations can rely on information stored by the SOR system for regulatory best-execution reporting, information obtained from the log files, or even real-time data intercepted using network capture. Calculations may be performed in Excel or using dedicated software packages.

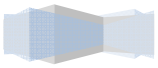
3.6 Operability Testing

This section discusses an activity that is rarely included within the scope of general quality assurance. The main goal of operability acceptance testing is to provide confidence to the system's business owners and Operations Support that a service/application:

- Is in line with general communications infrastructure within the company;
- Is well suited for automated maintenance procedures;
- Complies with existing monitoring standards;
- Satisfies approved resilience requirements;
- Follows existing security requirements;
- Corresponds to general technical standards;
- Has proven and tested support procedures.

Operability tests provide valuable insight into problems that may appear in production. The presence of QA personnel responsible for in-depth analysis of maintenance procedures helps other teams to address problems with test environment stability common to the testing of advanced execution platforms and other trading systems.

The following is a typical checklist for operability testing:



- Environment Documentation;
- Communication Protocols;

- Automated software deployment;
- Applications start/restart and Maintenance;
- Operating systems and Database versions;
- Monitoring and Health Checking;
- Debugging and Troubleshooting;
- Load balancing;
- Failover procedures.

Each test environment used for the SOR testing preparation or execution should be documented in the form of environment diagrams, access rules and support responsibilities. Such documentation needs to cover the output of operability testing and contain short and clear instructions for performing start, stop, deployment, clean-up, monitoring and log file processing.

4. VENDOR PROFILE: ALLIED TESTING

This section contains a description of Allied Testing tools and practices to provide QA support for advanced trading solutions.

4.1 About Us

Allied Testing is a specialist QA and testing firm with the sole focus on the capital markets, trading and financial industry. Allied is a global company with offices in the US and the UK and test labs in Eastern Europe and South America. Since 2000, Allied Testing has been delivering premiere QA solutions and services to leading global companies, including Thomson Reuters, Barclays Capital, JP Morgan Chase, RI3K, Equiduct Trading, and others.

The combination of the company's specialization, experience in testing complex trading systems and applications, the strong track record with global client base, and offshore/nearshore service delivery capability makes it a natural choice for QA and testing services to the capital markets and trading industry.

In order to handle complex quality assurance and development projects, Allied Testing has established a set of strong project management practices based on our prior work experience with leading financial institutions and our knowledge of such models as CMMi, RUP and MSF.

- Project initiation stage: establish communication, develop a test strategy; plan testing, establish tracking and reporting channels;
- On-going planning and tracking: weekly conference calls, weekly status reports, transparent cost planning and budget reporting;
- Deliverables and standards: agreed upon with each Client and adjusted individual Client needs;

Very often the effort required to create a comprehensive testing framework for an advanced execution platform is underestimated. Due to a large amount and variety of test scenarios, one needs to allocate a considerable amount of manpower to the task. As a result, the total project cost project becomes quite significant. Allied Testing addresses that by delivering much of the work remotely, at offshore rates.

Normally, clients of testing vendors expect to delegate repetitive, relatively simple and resource intensive tasks to an offshore provider whose personnel are ready to follow simple instructions. These expectations raise serious concerns about the offshore vendor's capability to deliver services for such complex systems as advanced SOR. Unlike many other vendors, Allied Testing offers quite a different service: we provide stable, motivated teams of young and dedicated Quality Assurance engineers. Our staff see QA as a long-term career choice, not as a stepping stone to development or management.

In order to match our Clients' schedule around the globe, we adjust our office hours so as to guarantee our European teams' 100% overlap with Europe, 85% overlap with the US East Coast, 65% overlap with the US West Coast, and 50% overlap with Australia and Japan. Our South American delivery center can guarantee 100% overlap with any US based client and 80% overlap with Europe.

Sometimes, a trading system is improperly documented; in certain cases the implementation comes ahead of its documentation. Allied Testing approach to these problems is to provide staff with sound English language communication skills. They are capable to fill in the gaps and obtain blocks of information from development and analysts. They can also assist with documentation update according to the actual implementation of the system, etc.

An advanced SOR system usually contains a tremendous amount of specifics. As a result, knowledge management becomes a very important activity. We are able to handle it thanks to our low attrition rates, the availability of backup for key specialists and knowledge preservation practices. Unlike an ordinary offshore vendor, the core team working on a project at Allied Testing becomes the extension of the Client's team.

4.2 BEST (Back-End System Testing) Platform

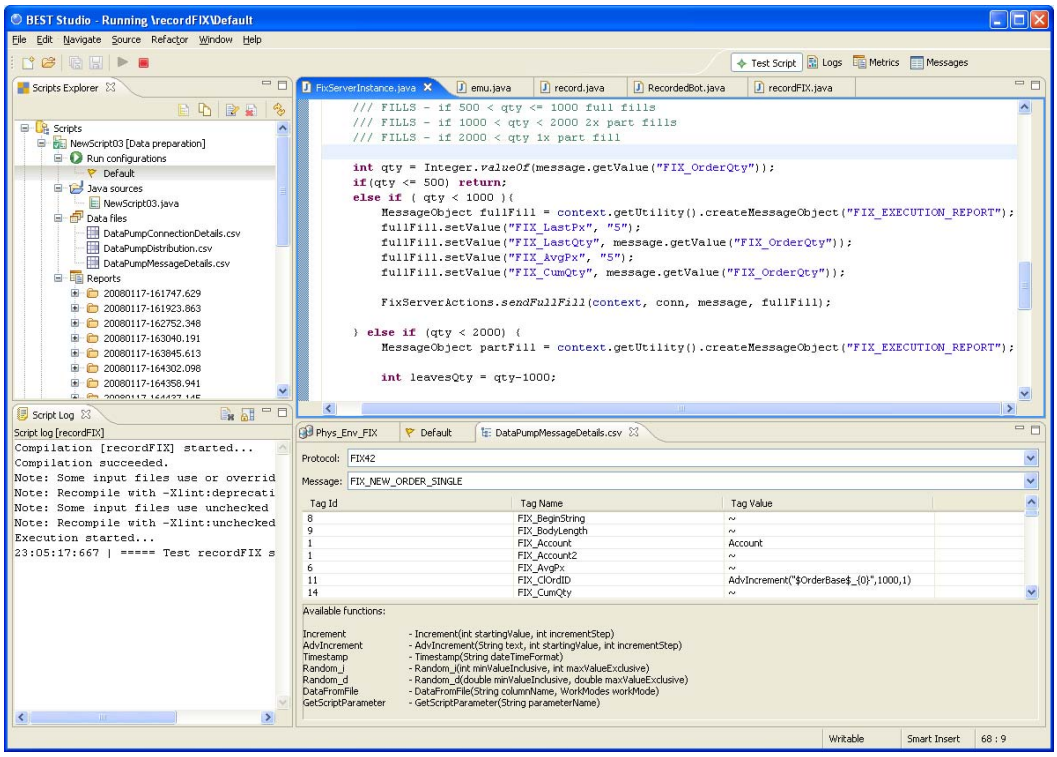
BEST (back-end systems testing) is a software platform for automated testing of enterprise applications and distributed server back-ends. It allows testers to develop and run test scripts performing various actions against servers and validating system response

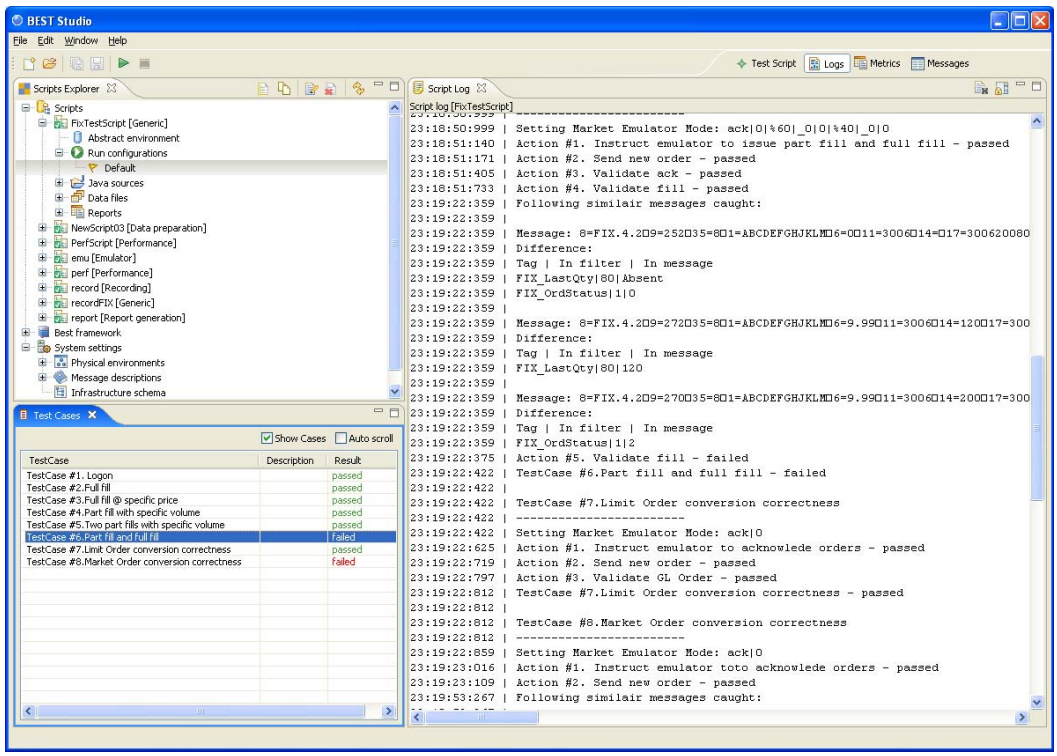
Our test tool is based on the Eclipse platform and supports all phases of the testing cycle: test recording, customization, execution and report generation. The test tool provides powerful scripting facilities and the ability to create script-less scenarios.

BEST provides unique performance testing and reporting features. The tool supports the following protocols:

- FIX 4.0 – 4.4 (client and server);
- FAST (client and server);
- Reuters RMDS (client and server);
- Fidessa OpenAccess (client and server);
- GL (the customer must have the corresponding license to use API);
- Generic TCP/IP, HTTP, SOAP, SSL;
- Message queues Tibco RV/EMS, JMS, IBM MQ;
- .NET Remoting (through a special agent);
- Integration with Mercury QTP to manipulate user interface;

- Database connectivity. KDB (server and client), JDBC ;
- Custom/additional interface can be added within 2-4 weeks if required.





4.3 Exchange Simulator

Allied Testing offers the Exchange Simulator product that helps algo, stat-arb and other systems-based traders test their intraday trading algorithms and strategies. It provides a convenient way of running and re-running strategies, using historical market data.

The Exchange Simulator reproduces all phases of exchange trading and forms a complete order book that behaves according to a full set of market rules. We convert historical market data into a constant flow of order submissions, amendments and cancellations, which are placed into the order book. All "user" or "external" orders are placed into a simulated book and processed according to the same exchange rules.

A comprehensive Market Model module constantly analyses the order book parameters and produces artificial events to simulate market responses to event flows generated by external users.

Exchange Simulator provides the following external interfaces:

- A UI for simulation management, locally or remotely;
- A remote API for simulation management from external applications;
- A trading interface (FIX 4.x or GL, custom protocol can be added);
- A Market Data interface (Reuters RMDS, custom protocol can be added);
- An Excel plug-in, which allows applying simulator developing strategies in MS Excel VBA; no need to integrate;
- Historical prices can be consumed in CSV format, Level 1, Level 2 or complete order audit trail data could be refined by our parsers.

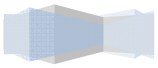
The following markets are supported by the Exchange Simulator:

- LSE (SETS)
- EuroNext
- Xetra
- OMX
- Milan
- Madrid

If requested, additional Exchanges/Markets or Dark Pools can be added within 2-4 weeks per execution venue.

4.4 Performance Analyzer

Performance Analyzer Tool (PAT) is a tool for real-time monitoring of trading performance. It listens to all SOR actions (order submissions, modifications, and cancellations) and exchanges notifications (acceptance or rejection of orders, execution notifications, etc). By comparing the parameters to current market data and benchmark values, it computes a number of metrics that characterize various aspects of trading performance:



- System metrics showing low-level performance of the trading client application: network delays, computation times, etc;

- Trade style metrics highlighting the particularities of the trader's (or algorithm's) trading style: aggressiveness, price prediction quality, etc;
- Profit and loss metrics showing how various factors contribute to the net result of trading compared to a chosen benchmark.

The metrics are displayed in a versatile, easily customizable grid interface with powerful grouping and sorting capabilities providing aggregate values and drill-downs to individual transactions.

PAT is meant to compliment the Simulator: it provides efficiency analysis of execution algorithms tested against the simulated data.