



+ Case Study

# Multi-language Exchange Matching Engine and Trading Platform

*An established exchange and their joint venture partner the needed a solution to allow the trading of cash and futures on environmental products.* The platform needed to support both English and Chinese languages and be integrated into existing clearing and product registry systems. The project had a short time line of six months from inception to delivery. Additionally, the IT staff of the client had day-to-day responsibilities that would prevent a delivery of the application in the required timeframe.

Connamara assembled a team of three full-time developers, a part-time test developer and a part-time business analyst/project manager to deliver the solution. **There was no** formal quality assurance staff assigned to the team. Quality assurance was implemented throughout the implementation by using agile methods of developer unit testing, end-to-end integration testing, continuous integration, automated application build, test and deployment.

## Why the Project was a Success

Connamara Systems, in six months, was able to implement and deploy a production ready electronic exchange and trading platform using only a staff of three full-time developers and two part-time business analysts. How was this possible?

### Focused Implementation Team

---

The full time IT staff of most organizations have day-to-day responsibilities that prevent the staff from completely focusing on the new project. Connamara, on the other hand, assembles teams whose only responsibility is to deliver a single project. Connamara developers are only assigned to one project team at a time. By the team only having one project to work on at a time, the team develops an esprit de corps that further propels productivity as opposed to wasteful silos of responsibility.

### Flexible Development Methodology

---

The Connamara agile development methodology allowed the client to change direction during the project. This is evidenced by the Chinese requirement that was not known at the start of the project but came to light about two thirds into the implementation. Another requirement added late in the implementation was the addition of support for futures trading. Because of the agile development methodology employed, neither of these new requirements caused a major disruption in the project.



## Domain Expertise

---

Connamara only works in the trading industry. This means that Connamara is well versed in the language, business and idiosyncrasies of trading - both from the perspective of end-users and providers of execution venues. This knowledge allows a Connamara business analyst to act as a proxy for the customer, shortening the time for communicating requirements between the customer and the development team.

## Requirements Verified by Integration Tests

---

Each high level, contractually defined requirement was verified by end-to-end integration tests. Each requirement was verified by an average of thirteen integration tests. The tests were implemented in English and Chinese.

Because the solution could be logically divided into an exchange matching engine and an end user trading platform, two types of integration tests were designed and implemented. First, standard “black box” messaging tests were created to isolate and verify the functionality of the matching engine. These tests would exercise the matching engine at the messaging API level. The Connamara TestRunner® would act as one or more client applications connected to the system and send in messages with the expectation of some known response.

An example test might be, Test Client A would send in a new order message to the matching engine and expect to receive a message acknowledging receipt of the order from the matching engine. Client A would also expect to receive a market data message indicating a change in the order book for this product. If a second test client (Test Client B) was also connected to the matching engine, a market data message would also be expected to be received. The Connamara TestRunner® would evaluate the actual messaging received by each test client versus the expected messaging as defined in the integration test. The Connamara TestRunner® would then record the results of the evaluation as “pass” or “fail”. Thus, if the tests that defined the requirement were passing, the requirement could be considered as complete.

The second type of testing used an open source testing framework to extend the scope of the integration tests to the trading user interface. Tests were written that would exercise the client UI automatically, as if a human QA tester was interacting with it. For example, a simple login test. The test framework would launch the trading client UI. Once the login dialog box was present, the framework would enter a valid user name and password. The framework would then validate that the user was authenticated and that the application launched in the expected manner. The test framework would then report the results of the test.



## Developer Unit Testing

---

Each developer at Connamara uses unit testing as a means to define and verify that the code they are creating is functioning as intended. By having unit tests around the code, the code behavior is documented and verified in a way that actually executes the code. This nearly eliminates the need for developers to spend time on code comments or standalone documentation that rapidly can become out of sync with the actual code.

More time writing code means higher developer productivity. Another benefit of unit tests (and all tests) is that they become a simple way of determining if a change will have unintended consequences elsewhere in the system.

## Test Automation and Continuous Integration

---

As described above, the platform was written concurrently with integration tests and developer unit tests. Utilizing test and build automation along with Continuous Integration reveals the true value of creating these tests while the code is being written.

Test automation allows all the tests that are defined for the system, both developer unit tests and integration tests to be run in a single batch. The automated testing was conducted in a separate controlled test environment that was established as a mirror of the production environment.

Continuous Integration (CI) ties it all together. The CI server monitors the source code repository for changes. When a developer checks in an update to the code base, the CI server checks out the latest version of the source code. The server builds the application. If the build is successful, the developer unit tests are run. If the unit tests pass, the integration tests are run. If the integration tests (regression) pass the build a success. An email is sent to interested parties with the results (pass or fail). Next, deployment packages are created for the various environments (test, staging, production) and saved for later use.

## Building in Quality *Not* Testing for Quality

In traditional development processes, developers write code and quality assurance staff check for bugs. *In this implementation, Connamara developers and business analysts created tests that defined code level behavior, system level behavior and user interface behavior **while the code was being written**.* This approach totally eliminated the wasteful practice of having a separate Quality Assurance staff looking for defects. Defects and missed requirements are discovered within fifteen minutes of each code change by the developers. Either the code would not build or a test that should be passing, fails.



This approach places a greater responsibility on the team. It requires the business analysts to precisely define the requirements through executable, repeatable tests and the developers to create unit tests. It also encourages and enables the developers to check in code often which keeps changes small.

As the project progresses, more and more tests are created, with integration tests describing and verifying the requirements, and unit tests describing and documenting the code. The benefit of regression testing is realized **with each code check in**. This makes changes to the system much less painful as unintended consequences are caught – **with each check in**. This minimizes the pain traditional development teams have when rolling out new versions. A new version of this application could be rolled out with each code check in as long as all the developer unit tests and integration tests pass.

## To Learn More

---

To learn how Connamara Systems can help you solve your trading technology problems

Visit [www.connamara.com](http://www.connamara.com)

- or -

email [info@connamara.com](mailto:info@connamara.com)