Enabling ETL Test Automation in Solution Delivery Teams

Subu Iyer
subu.iyer@yahoo.com

Abstract

Most companies today deal with lots of data. Typically, this data is in several databases and across multiple applications. Companies also have to work with data from vendors and clients. To integrate data from multiple sources, Extract Transform and Load (ETL) systems are used. These ETL projects are handled by Solution Delivery teams where each project has its own specific requirements and constraints. This makes it very hard for teams to build automated tests and processes around the ETL solutions that would help them consistently deliver high quality solutions on time and on budget. These and a set of outdated tools and practices make testing ETL jobs a time consuming and labor intensive manual process. The lack of automated testing also makes it very difficult to build and deliver incremental changes and this hurts productivity.

This paper describes how our team, the Quality Engineering and Specialized Testing (QuEST) team, is driving a change in the way teams build and test ETL jobs at Cambia Health Solutions. We are doing so by engaging with the Solution Delivery teams, setting up automated tests and providing teams hands-on training so they can continue writing automated tests. This paper will also talk about the benefits of building and growing the test framework organically in an incremental fashion. We'll see how our team was able to overcome initial challenges and improve our ETL testing.

Biography

Subu Iyer has been working in Software Quality Assurance for 10+ years. Subu believes in building quality upfront and setting up processes that help us deliver quality products. At Cambia Health Solutions, Subu works with teams to build automated tests and sets up automated processes that make teams more productive.

Subu has a Masters in Computer Science from the University of North Texas, Denton, TX. He co-owns PDX FitNesse and Test Automation Users Group to discuss FitNesse as a tool to drive development and improve productivity.

1 Introduction to ETL and ETL Testing

Extract Transform and Load (ETL) processes are done to integrate data from different applications and vendors. ETL is a process where data is extracted from a source system, transformed and loaded into a target system. ETL jobs import data from flat files into databases, copy data from one database to another or export data from databases to flat files. Before the import or export, there are usually a few data transformation steps required to derive the target data from the source data. Some examples of ETL projects at Cambia Health include importing employee data from flat files, copying claims data from one system to another and exporting member information from a database onto flat files. The challenge in these projects is to accurately move data from one system to another while maintaining or improving data quality. In some cases, complex data transformation rules make the implementation complicated. To add to these, typical ETL projects handle huge volumes of data and this means that these solutions have to be efficient and scale well.

The target applications depend on the data load for their functionality. In other words, testing of ETL jobs is very essential to ensure that the target applications continue to function correctly. Conceptually, ETL testing tests the functionality of the components built to perform the extract, transform and load steps. Testing these components consists of setting up test data, running the ETL process and verifying the actual results with expected results. Automating ETL testing is exactly the same – set up test data, run the ETL process and verify the results.

At Cambia Health Solutions, the Quality Engineering and Specialized Testing (QuEST) team, is driving the change to help Solution Delivery teams be more productive. Most ETL projects in our company lacked formal testing. We are changing this by collaborating with the teams on test automation, building a common framework for ETL testing and conducting training sessions.

2 ETL Solutions and Testing

Once the implementation plan is laid out, ETL development tools with their intuitive and simple graphical tools provide an easy way to build ETL components and work flows. ETL tools also come with a powerful set of data transformation capabilities. The graphical tools and built-in libraries make it easy to put together the ETL functionality. However, the lack of testing tools within the ETL development tool kit makes unit and functional testing of these jobs very difficult. In many cases, during the project planning phase, the testing effort is not fully understood and this leads to an inaccurate estimate of the testing tasks. The lack of test planning and the lack of formal tools result in the lack of systematic automated test processes. This reduces testing to visual inspection and spot checking of data.

With no automated unit and functional testing, any defect fix or enhancement can only be verified by manual end-to-end tests. Manual tests are time consuming, error prone and don't provide quick feedback about the product being tested. Manual tests can't be used as a tool to aid incremental development.

3 Challenges to ETL Testing

The concept is very simple, but when it comes to actually building automated ETL tests, there are a few challenges. Let’s go over some of these.

3.1 Testing Tools

Typically, ETL development consists of defining data components, mapping systems and setting up jobs using commercial software. These commercial development kits with their GUI tools make it easy to build complex ETL components quickly. The code for the ETL job is auto-generated by these development kits. ETL testing consists of setting up test data to test the logic in the ETL jobs, executing the ETL jobs and running SQL queries to validate results. In most cases, ETL jobs map a hundred or more fields from one system to another. Setting up test data is a time consuming activity. In addition, complex data transformation rules can make it very challenging to set up test data. The lack of tools to automatically generate good test data also makes testing a more time consuming activity than development.

3.2 Technical Skills

ETL testing requires a good set of technical skills. There is no visual front end for testing ETL as with most functional testing. While the user interface in target applications can be used for spot checking data, such testing is insufficient and ineffective. ETL testing requires a good understanding of the data model, SQL skills and scripting skills to generate test data.

3.3 Development Model

At Cambia, ETL development still follows the traditional waterfall model. Development and testing tasks are scheduled in sprints, but they are in sequence. In a typical ETL development cycle, the components are built in a sprint and tested in the next. Ideally, we'd like development and testing in the same sprint with testing providing immediate feedback about issues in the components and requirements. The lack of automated tests and no quick feedback loop between developers and testers make true agile, incremental development very difficult.

3.4 Manual Testing

As testing lags development by a sprint or two, defects are found late. Running manual regression tests following defect fixes can be very time consuming, but necessary. This manual testing activity also prevents the team from working on automated tests.

3.5 Change is Difficult

In general, humans tend to resist change and hide behind excuses like - “Our ETL is too complicated to test” or “We have no time to test” or “Our ETL testing can't be automated”.

4 Misconceptions about ETL Testing

There are many widespread misconceptions about how ETL jobs should be tested. Let us look at a few of these.

4.1 ETL Tests are Complicated

There is a common misconception that automated ETL tests need to reproduce the ETL job in test code. This might work in an ad hoc test when spot checking data. However, such test queries can get very complicated and unreadable. Using such queries in automated tests only leads to unmaintainable tests.

4.2 ETL Testing is Data Testing

Checking data quality is a good audit step, but this is not the same as testing the ETL functionality. This misconception might be due to the fact that testing starts after the ETL job has been built, run and the target data has already been loaded. Now the tests try to make sure that the target data is right. This might work as an audit step, but without full functional testing of the various components in the job, testing is incomplete. Most ETL jobs operate on at least hundreds of thousands of rows of data. It is impossible to validate all of this data every time the ETL job is run.

4.3 ETL Projects Don't Need Regression Testing

ETL projects are built as custom solutions to very specific problems. This leads to the belief that once they are built, they don't need to be revisited or tested for regressions. As a result, ETL projects rarely plan for test automation. However, there will always be changes to the requirements even after the product is released. There will always be defect fixes. And even if we magically built a perfect product, things around the product will change and such changes will require changes in our product too. Without automated regression testing, making any changes to the existing product is very risky.

These misconceptions make it very difficult for anyone to get started on ETL test automation. Two years ago, when our newly formed team engaged with ETL Solution Delivery teams, we realized that these teams were struggling with the basics of ETL test automation. We realized that our team couldn't possibly automate all tests in all the ETL projects. Instead we could pair with the ETL teams to write a few automated tests, build a test fixture and lay out processes for testing. Doing so would enable them to do their own test automation in the future and better equip them to build and scale their solutions in the agile world.

5 Benefits of ETL Test Automation

The benefits of automated testing are well known. Automated tests can be run easily and frequently identifying issues as soon as they happen. ETL test automation has the same benefits. The initial cost and effort to automate tests can be high, but over time and with an established framework, these tests can be implemented quickly and run cheaply. With these tests running in the background, the team can focus on more important issues like "how do we make our job more efficient?", "how can we extend this to work for files from other vendors?" or "how can we simulate and test error conditions?"

Automated tests provide a safety net for the development process. With these tests in place, developers can edit existing components for enhancements and bug fixes with the confidence that any new issues they introduce will be caught. Conversely, without the automated tests, it is very difficult for developers to make changes to existing components. The fear of introducing issues which go unnoticed due to the lack of automated testing is enough to deter anyone from making changes. Typically, this results in multiple copies of components, copies of components with subtle differences and components where defects never get fixed - a maintenance nightmare.

6 Enabling ETL Test Automation

At Cambia Health, the QuEST team works on building and improving processes. We focus on functional test automation and automated jobs to build, deploy and run these tests. These automated tests are one set of systematic Quality Assurance procedures that will help us deliver high quality products consistently on time and on budget.

The Solution Delivery teams, on the other hand, work on date driven projects to build new ETL jobs and enhance existing ones. On such date driven projects, the entire team is focused on building, testing and delivering the ETL functionality. It is difficult for such teams to switch gears and work on building better processes and improving their current practices. While these teams may realize that such improvements are very necessary for continued success, they often can't prioritize them within their tight deadlines.

Our team fills this gap - we collaborate with the Solution Delivery teams to design and develop automated tests, set up processes to build, deploy and run these tests in a fully automated fashion and help them learn and adopt these process changes that can make them more productive. We do this by pairing with one or more members of the Solution Delivery team to implement the changes. For example, when writing an automated test to verify target data, the Solution Delivery team identifies the target objects and formulates the query. Our team creates the test project, plugs the query into the ETL test runner and sets up automated test runs. We do this together for the first few tests. Then, the Solution Delivery team is able to add and update tests themselves. In other words, we enable the Solution Delivery teams to build automated tests and improve their processes. These changes reflect positively in the quality of their product.

6.1 Best Practices Exchange

Any time significant process changes are introduced, the biggest barrier is the resistance to change. We have seen this when transitioning from waterfall to agile development, when requiring unit tests for all new code written or when introducing the concept of using test cases as a substitute for detail requirements. We expected to face similar resistance to ETL test automation. In our initial discussions with the Solution Delivery teams, we were bombarded with questions on complicated edge case test scenarios where test automation wouldn't work, challenges posed due to out-of-sync test environments and insufficient privileges to systems under test. These issues definitely needed to be addressed in order to be completely successful in running fully automated tests, but were irrelevant at the outset. Our initial discussions went almost nowhere due to teams that were unaware of the benefits of test automation or had never seen successful test automation.

At the time, our company had a Software Quality Assurance Best Practices Exchange (SQA BPE). The SQA BPE is a forum to discuss anything related to software quality, demonstrate tools and processes or just talk about issues that need to be fixed. However, it was a closed network of managers and some senior quality engineers. We changed this. We opened up the SQA BPE to everyone, set up weekly discussions and published a public calendar of events and topics. Now, anyone in our company could set up a discussion, post questions or present their work. We used this forum to discuss problems seen with ETL testing practices, suggest possible solutions and demonstrate a prototype test automation tool. The wide audience included quality engineers, system analysts, developers and a few managers who helped us with productive feedback. Later, we presented automated ETL tests that we built with some teams to demonstrate success.

Our discussions do not always attract good participation, go smoothly or end favorably, but we do have a forum. We get to pitch ideas, participate in discussions and drive change. We realize that driving change is not easy and that it can take years for us to implement some changes and see the benefits of some others, but we believe that the change to adopt ETL test automation is very essential to increase productivity.

6.2 Build a Better Tool

Before we embarked on building a framework for automated tests and a test runner, we weighed existing options. We looked at add-ons for the ETL development tool we used at Cambia Health. We considered other commercially available ETL test automation tools. These solutions were either prohibitively expensive or required the use of proprietary techniques to automate tests. We realized that following these paths might require teams to learn another new and specialized skill. Such efforts are usually not cheap and typically don't scale well beyond one particular tool. We also looked at existing test automation projects. Both these projects required the adoption of a new platform for test development work. In addition, they also needed QA people to be proficient in coding. While our QA team members were very strong in SQL, they lacked the ability to program in languages like Java and Python. These in-house solutions were limited to the two teams that had QA members proficient in these languages.

We realized that any new tool we introduced would have to satisfy the following requirements -

- Getting started is easy. Automated tests can be built using SQL.
- No disruption to existing processes.
- Operating system and database independent.
- Tests would be available for everyone to read and understand. This would help all team members to participate in test development.

In addition to these, we recognized the need to build the tool in an incremental fashion. We did not have exhaustive requirements for this tool and we did not have the luxury to spend several months working on a perfect tool. We decided that building this tool incrementally was the best option. This method would allow us to try out features as soon as they are built, accept feedback from teams and grow the tool in an organic fashion.

We did not intend to build all the components of this new tool from scratch. Ideally, we would like to assemble a tool using a set of proven libraries. We wanted a mature platform and language that were widely used by development teams in our company. The advantage in picking a widely adopted language was that we could use help from other teams in our test development effort. We could also use all of the existing build tools, processes and infrastructure to build, deploy and run our tests. Since many development teams in our company use Java, this choice was easy. We'd use Java as the language to build test fixtures. These fixtures would have the ability to run SQL queries and interpret the results. The tests written by ETL testers would be in plain SQL which these Java test fixtures would execute. This way, ETL testers would never have to write or even look at the Java code. Java also has a good set of libraries for database access which is really the most important feature of this tool.

The other major requirement was the ability to share tests with the entire team. Every member of the team would be able to read the tests, run them and interpret the results. This would let anyone on the team run tests at will. This also freed up the QA members to focus on more important tasks rather than running existing tests. We looked at a few test runners like Cucumber, Robot and FitNesse. All these test runners let us define the tests in plain text which make them readable. In the case of Cucumber and Robot, these tests are in text files in the file system. FitNesse, on the other hand, serves these tests as wiki pages. These tests can be easily accessed from anywhere using a browser. FitNesse tests can be run from anywhere by clicking buttons on the wiki page. FitNesse is a wiki and a test runner. Now everyone on the team can easily access the tests using a browser, read them and understand them. Once we decided on Java and FitNesse, the next step was to write a fixture class in Java.

6.2.1 Test Fixture Development

Our mantra here is, “Start simple, start small”. The reasoning behind this was that we had seen many test framework development efforts fail. These failed efforts attempted to build the perfect tool upfront and the end result was a tool with a clunky interface and proprietary syntax that no one wanted to learn. We kept the design simple and our code was simpler - just enough to make the first test work. Our first test connected to a test database, ran a query and compared the results with an expected result set (see Figure 1). We were able to whip up this functionality in a couple weeks and demonstrate it at the SQA BPE. By presenting this functionality, we wanted to let people know that they could try out simple tests against their databases by writing plain SQL. We also wanted to gather any feedback from them to continue adding functionality to the test fixtures.

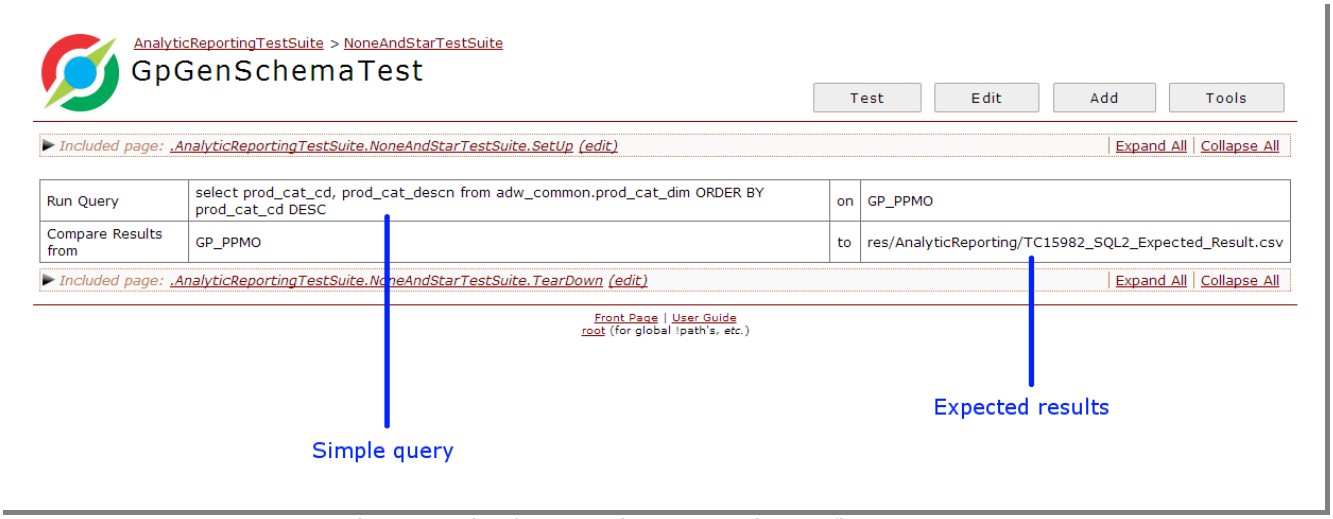

Figure 1. Simple test using a generic test fixture

The most common criticism we got was that the tool and the testing concepts were too simple and would never work for real ETL jobs. But the simplicity was its strength - simple things work well, they grow and scale easily. A couple teams understood this concept. They wanted to know more about the tool, build a few tests themselves and understand how they could use it in their projects. We were not ready to hand over the tool to these teams. We decided instead to work with them to build their own tests. We paired with the QA members of the team to write the FitNesse wiki tests with the SQL queries in them. As the test queries got lengthy, we added a functionality in the test fixture to read queries from a file. So now, the wiki tests read lengthy, complex queries from files (see Figure 2). Logical file names for the query files also made it easy for other members on the team to understand the intent of the query.

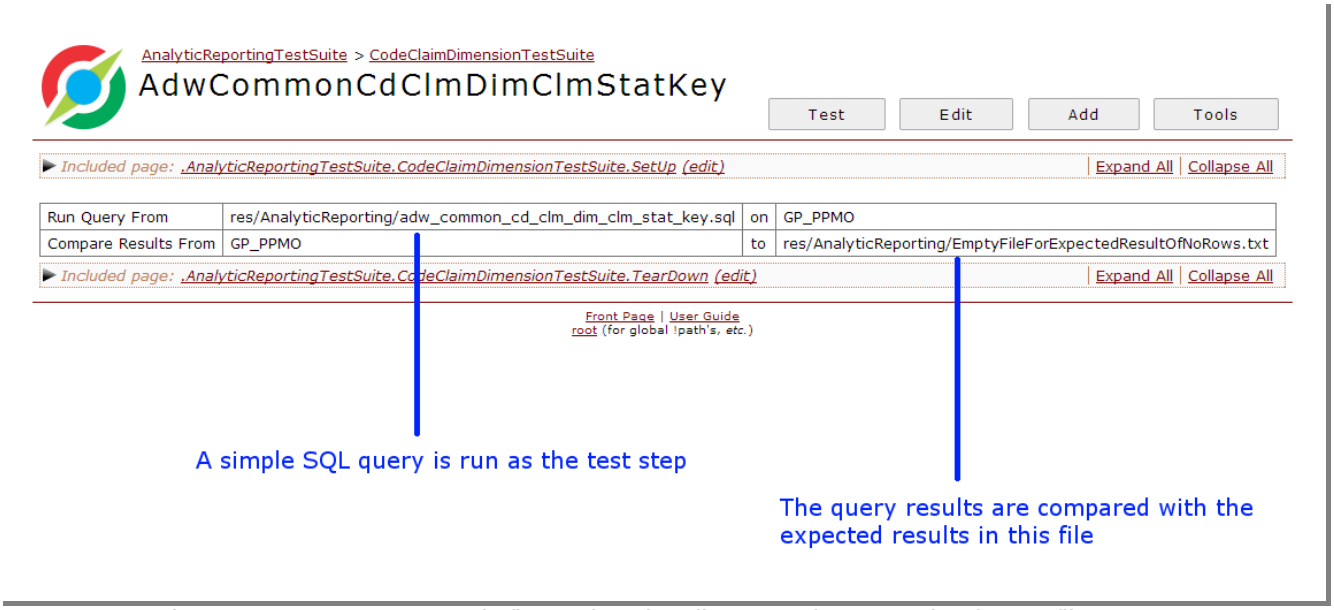

Figure 2. Test uses a generic fixture functionality to read test queries from a file

We have also recently developed tests with a product owner. These tests are more readable and express the functionality in plain English (see Figure 3). We were able to achieve this by extending the base functionality and adding a few wrapper methods to the fixture code.

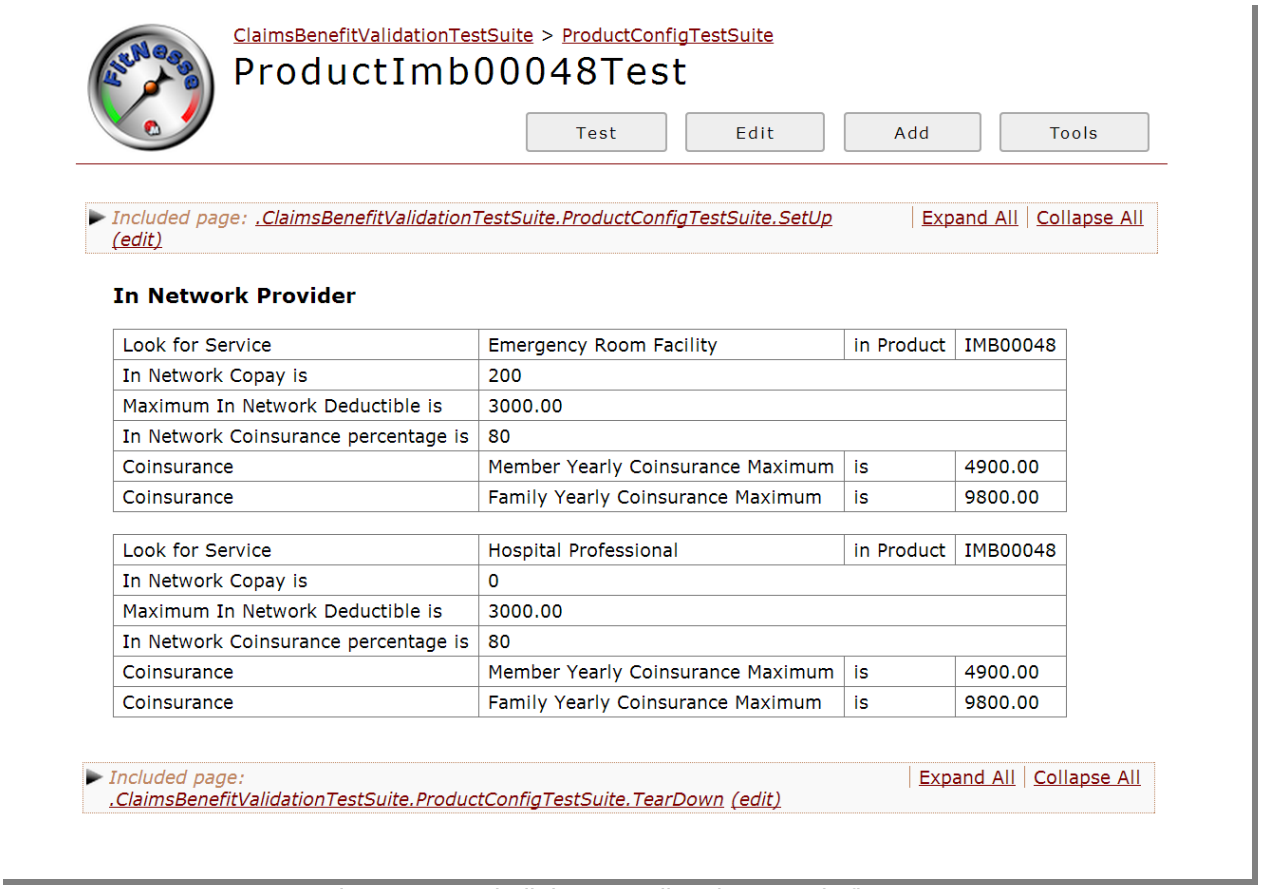

Figure 3. Test built by extending the generic fixture

More than a year later and with five teams using this ETL testing tool, the core functionality of the tool remains much the same - connect to a database, run a query and compare the query results with a set of expected results. We have added some utility methods and helper classes, but the core functionality remains the same. This shows that testing tools and frameworks can be built successfully starting with just enough functionality and adding more incrementally.

6.3 Educate

Educating teams about the benefits of test automation and the need to constantly improve existing processes is an ongoing task. So is learning the needs of the delivery teams to be able to build better tests. There are many opportunities to learn from each other, improve the tools we build and our skill set. So it is very important to engage with the teams and continue discussions.

Education can be done simply by sharing success stories at the SQA BPE or by holding elaborate training sessions. We are doing both. By sharing success stories, teams get to see and hear how others use the ETL testing tool in their projects. This acts like an endorsement and also helps people see similarities between their projects and advantages of using a common tool. At the time of writing this paper, our team had completed a two part training session on how to use the ETL testing tool.
We conducted workshop style training sessions on the following -

1. Introduction to Test Automation where we covered the basics of the FitNesse wiki, reading tests, running tests and editing them. Attendees worked on a sample test project we built for this workshop session.
2. Basics of ETL Test Automation - here we went over examples of tests that ran queries against a test database.

We are planning to continue delivering the same sessions a few times a year. We are also planning to deliver at least one more course on advanced ETL testing that will focus on extending the existing framework.

7 Some other Simple but Effective Techniques

Along the way, we discovered several techniques that are natural, but aren't really obvious until you do them. Simple things like informal discussions over coffee, celebrating successes, talking through pain points together go a long way in finding solutions and building collaborative relationships. Similarly, telling people they are wrong and pointing out flaws are not constructive at all. This seems obvious now, but we learned it the hard way.

8 Conclusion

Two years ago, the QuEST team started this focused effort to improve the quality of ETL products. We recognized that lack of automated testing and lack of skills to build automated tests as areas that needed significant improvements. Our work on building automated tests and a framework for testing have enabled teams to build automated tests and making these tests part of the development process have shown good results. Below are some preliminary results -

- Five out of the eight ETL teams have some automated tests. These tests are scheduled to run daily. They can also be run by anyone on the team by clicking a button. On one particular project that generates member benefits documents, testing consisted of end-to-end tests with visual

inspection of documents. This was a time consuming process and defects were found very late in the development cycle. Now, automated ETL tests have been implemented to validate member data loads and document configurations. Issues are now found as soon as the ETL jobs have finished running and are quickly fixed. These automated tests have also drastically reduced the time and effort to run tests. Previously, the testing cycle took 4 to 6 weeks and followed development. Now, many tests are run in parallel with development and the final end-to-end tests can be completed in one week.

- Setting up automated tests is now easy and quick. Teams know how to get started and how to get help. Recently, a team was able to set up automated tests for almost 60% of the test cases even before the functionality was built. These tests caught defects, issues with the environments and identified gaps in requirements early.
- We have had a few enhancement requests to the ETL test runner from System Analysts. One such enhancement was to display the query run in the test. Looking at the query would give them more confidence in the tests. A few Product Owners are thrilled at the prospect of using FitNesse for test automation because they now understand exactly what the tests do. We now have non-QA team members participating in testing activities.

ETL test automation is real and can be done with simple tools and techniques. ETL test automation can be implemented in an agile fashion – one can start with simple tests and over time add tests to build a comprehensive test suite. ETL test automation, much like any other functional test automation, helps development by finding issues as soon as they happen and tightening the feedback loop. Test automation can be used to drive the development of ETL components. While it was a struggle to get early adoption from teams, we have seen good progress recently in teams accepting a common framework and embracing sound testing concepts. We believe that over time we'll continue to hone our test runner, improve tests and test coverage and these changes will result in a better product.