De-Mystifying Data Testing and Applying Automation

Pad Balasubramanian
Senior Principal Consultant
Center of Excellence Team
Syntel Ltd, Chennai, India

Prasanth Malla
Automation Architect
Center of Excellence Team
Syntel Ltd, Chennai, India

ABSTRACT

Being Agile has become a norm rather than a special need. To stay Agile in today's world requires significant thought and innovative solutions. The testing industry has matured during the past decade with hundreds of open source tools and frameworks, specifically in the area of automation. QA teams have significantly benefitted by this evolution, enabling them to satisfy the demand of being Agile throughout the lifecycle and stay at par with technology advancements.

One of the most important objectives of data-testing is to recommend the corrective measures the back-end integration teams need to introduce in the development life cycle (SDLC). Data validation definitely plays an important role and there are lots of techniques and tools available in the market. However, end-to-end automation penetration is comparatively low in back-end data testing and ETL test automation since the data transformation predominantly happens through ETL processes on major enterprise systems. There is a clear market and industry demand for automation in data testing. This space is gaining importance with the sole reason being quantity (size) to be handled along with the quality of data.

This paper explains the essentials of data testing strategy - how data quality and data validation checks play an important role; where and how to bring-in automation; and finally the method for arriving at faster, accurate root-cause analysis. It can be argued that data quality checks are implicitly covered as part of validation, however it is always recommended to address the problem at the source rather than at the destination. According to analyst findings in public domain, significant revenue wastages are reported due to poor data quality.

The approach defined in this paper will benefit QA-testing teams involved in back-end data testing. It will improve their understanding and enable them to apply correct techniques as they move forward. Automation for data-testing is considered only for people with a technical background. A proper understanding of what exactly happens at back-end once data is processed from front-end, will enable a non-technical person to understand, enjoy, and appreciate the benefits of automation.

Keywords

Data quality check, data validations, ETL test automation, Agile development, third party systems, data source, and data destination.

1. INTRODUCTION

"Upstream inefficiency or issues induces defects in downstream systems and applications." Development teams spend more time and effort to identify the source of the defect rather than fixing it. To understand this statement better, let us go through the below illustrated example:

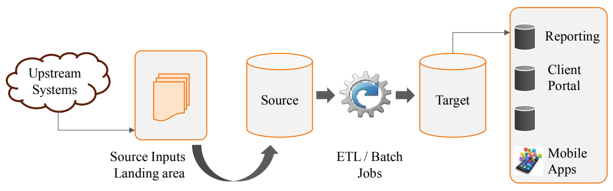

Fig 1: Back-end Data Transfer Workflow

In the above example, defects at the downstream systems like reporting and client portals are typically attributed to factors at the respective application layers or the ETL batch jobs. There is yet another key cause.

Typical data-testing happens between the source and target DB. For instance, if there is a valid defect in this process, the initial reason can be attributed to the ETL batch jobs that moves the data from source to target. However, there could be a scenario where the actual root cause could be at the data origin - source of the data originated from the upstream systems.

A defect is a defect regardless of the cause. It is important to have a mechanism to identify the root cause in a short time frame to get an instant fix. This can be achieved only if the approach of the testing strategy, specifically on back-end data testing, is well thought through taking into consideration the data flow between various systems. The below sections explain the key steps involved - where and how automation can be introduced in the overall data testing life-cycle.

2. IMPORTANCE OF SOURCE DATA CORRECTNESS

Data testing should not be assumed as data validation testing or data cleansing. A relook at fig 1 shows that the first step should be to ensure data correctness of the input source data. Large enterprise systems, especially in the banking and insurance domain, have heterogeneous data source formats: csv, json, xml, excel, and .dat. Large enterprises also have huge volumes of record-sets and sizes. Moreover, the format and the content structure are governed by the rule-dictionary agreed upon between the systems. The origin source can be from a third-party provider or an in-house application/provider. A typical example being daily credit-card transactions summary statement. In an ideal business scenario, the systems involved in sending and receiving have a common data-exchange protocol or rule dictionary.

However, in the current agile world, no situation can continue to remain "ideal" as changes are bound to happen and it can occur anywhere. It cannot be assumed that the input data-source is 100% accurate at all times. This statement is even more valid in pre-prod environment. There can be several external/internal factors due to which there could be deviations in the source data with respect to the pre-defined format dictionary. The current trend in automation for ETL-testing is more towards the final phase of the journey which is data validation. It will be interesting to see how automation can be leveraged for earlier phases that is, data-quality checks.

2.1. Automated Data Quality Check

It is indeed very demanding to design a system to handle the heterogeneous data sources. With the advent of complex online systems, e-commerce transactions, and so on, the diversification is multifold. There are no standard bodies to govern the structure of business data. Organizations define their formats depending on the technology of their IT landscape. The source can vary from industry standard XML to legacy text file formats. Hence any attempt to introduce automation must consider these facts and be adaptable to scale.

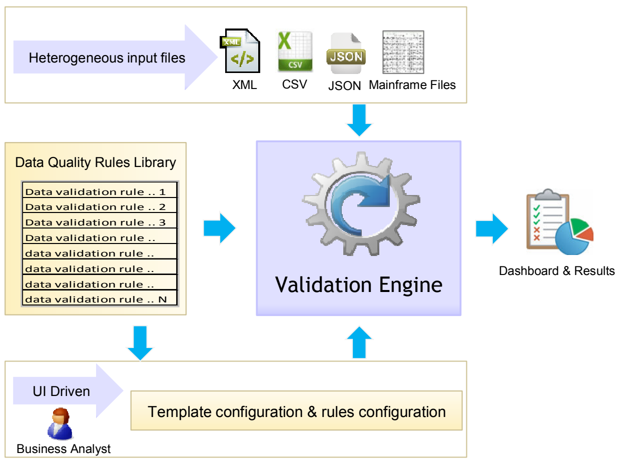
Heterogeneous input files
XML
CSV
JSON
Mainframe Files
Data Quality Rules Library
Data validation rule .. 1
Data validation rule .. 2
Data validation rule .. 3
Data validation rule ..
data validation rule ..
data validation rule ..
data validation rule ..
data validation rule .. N
Validation Engine
Dashboard & Results
UI Driven
Business Analyst
Template configuration & rules configuration

Fig 2: Automated Data Quality Checks

The above figure explains the process with a rule-configurable validation engine, enabling automated data quality checks. The validation rule engine is configured with the expected data-content's range/type/formats. The configurations can be in multiple sets, as the inputs can come from different input streams or systems. The engine should be able to accept heterogeneous inputs as described earlier. Some of the sample parameters to be considered are listed below:

[1] Fixed validations: Transactions should have few mandatory fields like originating party, third party, transaction date, amount, currency, transaction ID, and values for the same to be in line with the defined format value.

[2] Validation with external references: Transaction charges should be subjective to the third party based on the transaction amount and transaction type.

[3] Contextual reference validations: A field within a given record can have dependency on another field in the same record. For example, in case of a card transaction payment, the card number field is mandatory and the card number field value should be as per the definition. In the event of the transaction being an online direct transfer mode of payment, the card number field could be optional. The card number field validation can be subjective in some cases as well.

In addition, format checks are other important validations to be done on the input data records.

3. SOURCE TO TARGET DATA VALIDATIONS

For any data movement between two systems, it is mandatory to carry out source versus target validations. The data movement can vary from a simple DB transfer to complex transformational migrations. In addition, there can a single stage migration or multistage data migrations. The role of the quality assurance teams here is to analyze and understand the process of migration and apply appropriate validation principles as part of testing. This is the key for a successful test completion.

In large enterprise applications, data migration can include large volume ranges that vary from a few thousands of records to few millions. Quality assurance teams typically apply random sampling data validations, as it requires significant time and effort for a full-fledged source versus target comparisons. There are several approaches to address this problem utilizing open source tools as well as licensed tools. There are a number of articles and papers written on data validation approaches which can be referred to.

4. DESIGN OF FRAMEWORK - CONSIDERATIONS

The first and foremost thumb rule for any independent QA testing team is to design and draft test scenarios from the system requirements, that is, mapping rules. QA teams should apply caution while reusing the ETL tool's query output which is used for data migration post transformation. The simple reason being, a verification process approach should be independent of the creation approach. This ensures that in case of any defect the root cause of the defect might be due to the transformation logic during implementation.

Enabling Automation:

End-to-end automation on data-testing requires the enablement of automation on the following activities:

[1] Data quality checks

[2] Source versus target validations

[3] Defect analysis – leveraging the defect log data storage repository

Automation of data quality checks have been witnessed earlier. In order to enable automation on data validation, an appreciation of how ETL transformation works will immensely help. All ETL processes are designed based on mapping rules. The mapping rule set acts as a source for the ETL developers. To understand better, this is similar to the requirement / functional specification document for application development. Functional testing teams use these documents as a source for test design. QA team involved in ETL testing, should treat the mapping rule document as an important source for analysis and test design.

The first step would be to create SQL queries based on the mapping rule document. Depending on the complexity, the query is designed. The next step would be to execute the query against the source and target. If the transformation happens on a multistage phase as per map-reduce then the source target will vary accordingly.

The final step being validation of the retrieved values from source and target. There are approaches and tools available in the public domain for source-target validations. To achieve end-to-end automation, all the three steps have to be considered for automation, thus bringing in higher efficiency.

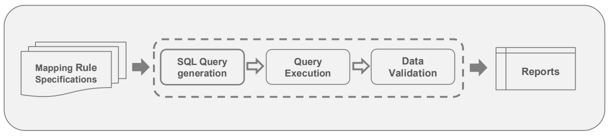

Fig 3: Enabling Automation for Data validation

To summarize, automated data validation involves tool enabled activities for the following:

[1] Conversion engine - mapping rules to SQL queries (assuming source and target are SQL complaint systems)

[2] Execution engine

[3] Validation engine

There could be a situation where the source and/or target may not support SQL operations and for those cases alternate approaches need to be considered for data retrieval.

Any framework involved in automation is expected to automate the said activities, seamlessly so as to enable continuous integration as well.

5. DEFECT ANALYSIS

Any testing activity is considered incomplete without a proper defect report filing and the subsequent analysis. Data-testing is no exception to that and it is even more important considering the complex heterogeneous data types, formats, and input sources.

5.1. Where is My Defect?

An accurate defect root-cause analysis immensely benefits the development teams significantly enabling the SDLC to achieve high degree of maturity and effectiveness. It is a proven fact on systems running for years that the time taken to identify the source of the problem is much higher than to fix the same.

Table 2: Types of testing corresponding to flow of data

	Data Entry	Data Exit	Application Layer
Data - Quality	✓		
Data – Validation		✓	
Functional Test			✓

To arrive at a quicker resolution, it is imperative to perform appropriate checks as illustrated above. As has been seen, data-testing is not a single step activity. At the point of arrival of source data, the data quality checks identify the defects which otherwise could pop-up in some form during the validation of the downstream applications. Data validation post migration helps to unearth defects due to the migration process.

The types of testing to be performed against respective sources to give a high-level of understanding:

Table 1: Types of testing on various sources

Sources	Types of Testing
Upstream Data	**Data Quality Check**
Migration / Transformation Process	**Data Validation Checks**
Reports Validation	**Reports Functional testing**
Downstream Systems	**Functional Testing**

5.2. Approach to Analysis:

One of the best practices as part of defect management is to enter the cause of the defects as part of defect fixing in the defect management tool. Teams use this information to create inference reports on the factors influencing the defects and take corrective actions.

Similar approach alone may not be sufficient for data testing, as many a times the fix could be a temporary one, to handle an incorrect entry from the data source. It is an acceptable business practice for project teams to have a quick fix and later go in for a permanent fix purely from a business perspective. A structured defect analysis comparing the relationship between the various stages in the data movement life cycle is required to enable development teams reach the source of the problem faster.

Let us consider the following:

[1] Set A: Defect catalogue captured due to data quality checks

[2] Set B: Defect catalogue at data-validation checks

[3] Set C: Defects observed at the reporting application layer

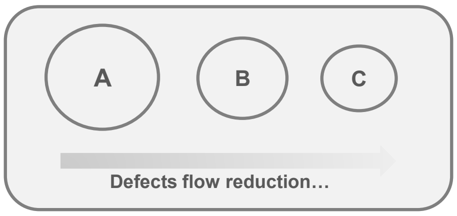

Fig 4: Defects reduction through continuous testing

By having thorough data quality and data validation checks, it is natural to expect a lower defects ratio at the reporting application layer. In the above approach, maximum data-source defects are blocked at the initial level, further at the data validation layer, and finally at the reporting application layer. Thus, defect reduction can be obtained through the process of continuous data testing at different layers.

Defects reduction in subsequent phases is one of the most important objectives of quality assurance teams. In this journey, it will be interesting to see how a thoughtful inference from the defect analysis can help to achieve defect prevention.

Let us elaborate on the above example. Due to factors like incomplete data quality checks, test-miss, and open/known issues there can be a situation where part of set A can be observed along with set B. Similarly, part of set B with set C, and finally both set A and set B with set C, as depicted below.

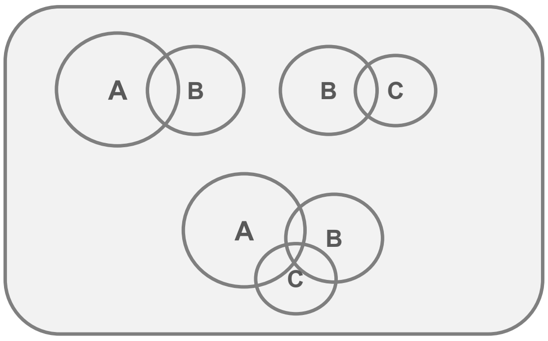

Fig 5: Defect analysis inference

The above three scenarios to be read as follows:

[1] A ∩ B – Common defects due to data quality issues and data migration, to be considered as high priority. There could be a dependent data-set within a same record not being handled properly.

[2] B ∩ C – Migration defects leaked to reporting (presentation) layer.

[3] A ∩ B ∩ C – Common defects carried across the stages, require fix at data source level.

Through the above approach, at the individual steps, project teams can take decisions on where more time needs to be spent for testing activities. By further applying additional cardinal set-theory concepts similar to above, more interesting inferences can be obtained which will help to achieve defect prevention for the subsequent cycles and releases. In addition redundant defects can be reduced.

There could be situations where independent QA teams might be involved on the above activities. However, by performing the defect analysis from the data origin through a structured approach, project teams' benefit on prioritizing the efforts as well as overall cost optimization.

A developer fixing a defect at reporting application layer, should know the origin source for the same in order to have a quick fix. For faster identification, the defect-log data storage repository plays an important role. As the data testing process moves from one step to the next, the defects summary log should be catalogued against respective processes.

6. NEEDS AND BENEFITS

The complexity of defect sources increases as the testing activity progresses, as represented below:

Table 4: Defect root cause classifications

Defects @	Possible Causes
Data Loading	**Data quality on upstream sources**
Data Validation	**Migration process and/or upstream data**
Reports Testing	**Upstream data sources, Migration process, Reporting process**
Downstream Applications	**All of the above in addition to application specific issues**

Imagine a situation of complex business systems, where the possible causes for a data defect on downstream systems could be anything from a simple coding error to reports to migration processes. It could be a developer's nightmare to analyze the root causes going by the nature of the multistage data movement. It is imperative that appropriate testing needs to be carried out by the teams involved in data testing at appropriate phases. Failing which, the amount of effort to be spent on issue resolution will multiply significantly.

Testing teams should be aware that data validation primarily checks source versus target. Hence in a pass-through migration, an invalid data at source could get moved to target as-is. Data validation checks will show as pass, however, downstream system will end showing a defect due to bad data. Hence by tracking the defect deduction metric at each phase, prediction analysis improves the subsequent testing phase. This also helps the release management teams in the decision making process.

Defect management is a complex activity on SDLC, especially when the release time period comes closer. The defect management process should not be seen only as an activity to achieve defect closure. In-depth inferences can be made which can help the projects teams in decision making. Data testing being a multilayered approach, a well-planned defect management process in the current release cycle, can help in preventing defects in the subsequent cycle by recommending appropriate corrective measures.

Consider an example, where a data defect was observed at the downstream systems. There can be several possibilities as to whether this defect was captured earlier or not, based on which project teams shall be able to apply measures at the right place. The below table summarizes the possible combinations while performing a casual analysis for data testing.

Table 3: Defect analysis backward tracing

Data Quality	Data Validation	Inferences
Defect	**No Defect**	**Migration / Transformation logic to be re-looked at.**
Defect	**Defect**	**Defect leakage, open defects.**
No Defect	**Defect**	**Possibility of data-quality rules not reflecting the actuals.**
No Defect	**No Defect**	**Indicates the upstream data source rule dictionaries are not updated; Upstream systems to re-validate the data rule dictionary**

For a given data defect at downstream applications, the associated defect should be tracked backwards as part of the root cause analysis. There can be complex combinations in this process. A careful and smart analysis can help achieve preventive measures as well as accurate root cause analysis:

[1] Defect observed at data quality check stage and not during validation stage – the migration/transformation process to be reviewed.

[2] Defect was observed at both data quality and data validation stage. Ideally at downstream this should have been a pass. Possible causes could be due to leakage or incorrect test and error due to dependent data mismatches. At validation stage, only individual records will be tested, whereas at downstream stage the record will be grouped as per business rules.

[3] No defect was observed at data quality, but defect was observed at validation. This can happen primarily when the rules at data quality are outdated, and the migration could be a straight pass-through without transformation.

[4] Finally, both data quality and data validation were passed, however, there is still a data defect downstream. Potential reasons could be that the upstream system still follows old rules and the same is reflecting at data quality checks. Both upstream systems as well as internal systems to reconfigure their rule validations.

There could be a situation where teams will be working hard to fix a defect, without realizing it is not a defect, rather an issue with incoming source from an upstream system which can be a third party service provider. A structured analysis helps to arrive at a conclusion quickly.

7. APPROACH FOR LARGE DATA VOLUMES TEST

One of the challenging tasks for testing teams involved in data testing is how to handle large data volumes, known as Big Data. Here the volumes can be multi millions of transactions. As explained in the earlier sections, appreciation about the migration/transformation process will immensely help to create a strategy. Without which, one could be lead to a complex, redundant cycle.

[1] For data movements of huge volumes through system driven commands record-level comparisons are redundant. Dimension level checks like aggregation and count are faster approaches.

[2] If the migration is tool-driven (any market ETL tool), an analysis should be done to check whether the movement is a pass-through or any other transformation logic is involved. A careful study of the mapping rule document will come handy in such situations. Teams can fine tune their comparison logic restricting to actual transformations rather than proceeding with record level validations for the entire set.

To summarize, large volume comparisons is very much possible and automatable, however it comes with a cost and effort. Enough care should be taken, as the data movement itself might take a few hours for, say, 10 million transactions. It will be redundant to invest equal or more amount of time for validation without analyzing the opportunity for errors. More the transformation logic, more the opportunity for defects. Straight pass-through means lesser the opportunity for defects. Testing teams should spend more effort where opportunity for defects is high.

8. CONCLUSION

To perform functional testing, there is a physical entity in the form of UI where the system tester acts upon. It is easy to learn and replay. However, for data testing, in the absence of user interface, testers are expected to have the basic understanding of the steps and processes involved. To be a successful back-end tester, an appreciation of data quality and data validation, process involved on data transformations are inevitable. One-click automaton which was possible for functional testing, may not fully fit data testing. However, a combination of individual tools and automated steps will bring in significant acceleration to the data testing journey.

9. REFERENCES

[1] The Six Principles of BW Data Validation. Sapiex White papers: http://www.sapiex.net/sapiex/sapiex.nsf/0/272DE7600522A3CE862578230056F4FA/$FILE/Sapiex_White_Paper_-_The_Six_Principles_of_BW_Data_Validation.pdf

[2] General concepts of Set Theory

[3] Yuan Wang, David J. DeWitt, Jin-Yi Cai, “X-Diff: An Effective Change Detection Algorithm for XML Documents”, http://research.cs.wisc.edu/niagara/papers/xdiff.pdf

[4] Parsing Techniques - A Practical Guide: http://dickgrune.com/Books/PTAPG_1st_Edition/

[5] XQuery/XML Differences. Available at: https://en.wikibooks.org/wiki/XQuery/XML_Differences