Healthcare Hardware Reliability Reporting System

Katherine Iaquinto
August 2024

This version of the deck contains more details in the following note sections

About Katherine Iaquinto

Master student in Analytics in interdisciplinary program
- Python for data analysis
- Visualization and front-end development

Data analyst at Hiya with responsibility to maintain and extend data infrastructure for reporting on mobile carrier and customer data
- Cloud data engineering (AWS)

Data engineer at T-Mobile
- Web application for reporting
- Power BI dashboards

See resume for more information including dates of experience

Project Goal

A healthcare product manufacturing Quality Engineering organization aims to better monitor the health of hardware modules during manufacturing and assembly.

This organization requested a design and implementation of a new reporting system:

- A scalable, secure, easy to maintain data pipeline that keeps current with automated loading of data.
- An interactive visualization dashboard **enabling the ability to visualize test results as a time series:**

Visualizing the test pass rates and overall trends

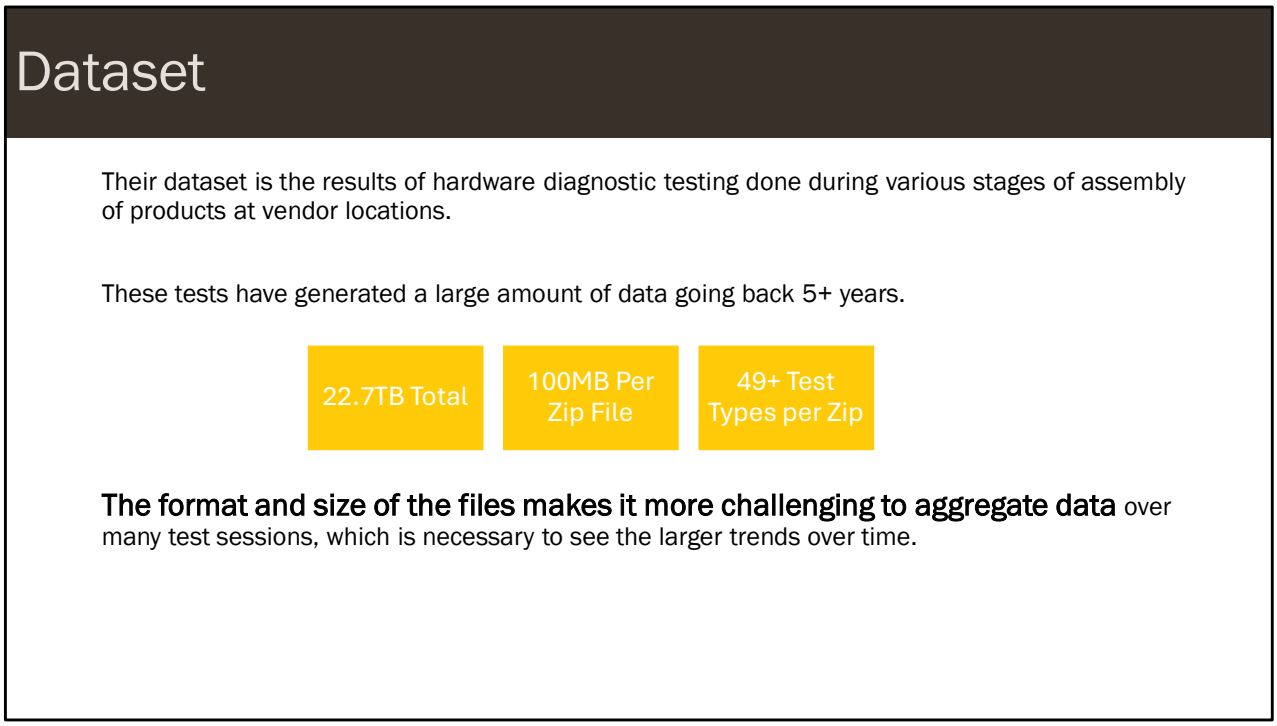
Enabling engineers / analysts to monitor specific metric values over time

The Quality and Reliability organization's goal is to create analytics to monitor the health of hardware modules during manufacturing and assembly. Visual analytics were specifically requested for detecting the trend in module pass/fail rates, as well as plotting the values of specific metrics measured to monitor their conformance to upper and lower limits. On the rare occasion that manufacturing defects are found, the visual analytics need to be interactive and responsive so that novel combinations of dimensions can be selected on the fly to make comparisons across time,

vendors, manufacturing stations, and more. This called for a redesigning the dashboards with charts that can zoom into and out of various time scales and have ability to be sliced / filtered a variety ways.

The business unit requests an enhanced reporting system:

1. A system that can be kept current with an automated loading of data.
2. A system that scales with data growth.
3. An interactive visualization dashboard template that can be used to make new dashboards that both answer current business questions and can be adapted easily for future questions.



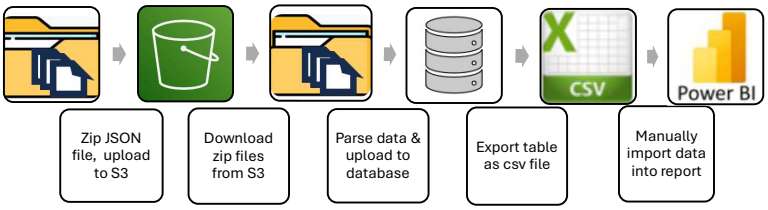
Quality team has been gathering the results of functional testing done during various stages of assembly of products at vendor sites for more than five years. Each functional testing session generates a large number of JSON files, which have been zipped into one zip file per testing session and uploaded to an Amazon Web Services (AWS) S3 bucket. This S3 bucket, *ultra-mfg-prod*, has approximately 23TB of data at time of writing. The bucket is divided into subfolders by vendor site name, which are subsequently divided into up to 9 subfolders 01, 02, 03, etc., corresponding to the

manufacturing test station where the tests occurred. These subfolder are further divided into subfolders which can be named after the serial number of the machine used for testing and/or the date of the test. As the files are buried in this deep structure and are not already in a format that is readily accessible for processing by visualization software, the need has existed for a long time for a system of parsing this data into a format that can be used to visualize the data for insights.

Previous Data Pipeline

Initially the project scope was to build a revamped version of the last 3 steps of the existing data pipeline diagrammed below.

Previous Data Pipeline



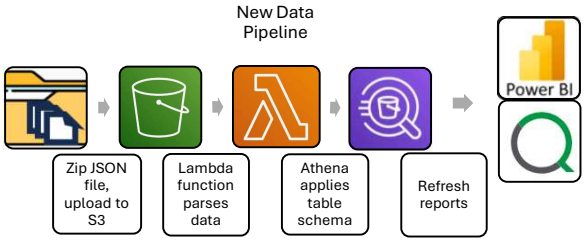
- Interviews with the team revealed that the biggest challenge preventing success with the **previous data pipeline was earlier, when downloading the files from AWS.**
  - This procedure was time-consuming.
  - Space on the server was a bottleneck.
  - Data on the server might not be backed up.

A previous data pipeline had been established, which is summarized in the process diagram below. From a lean process analysis standpoint, the previous process generated a lot of waste. The previous process contained several steps that had to be initiated by a person running a script. So even though there existed scripts to semi-automate these steps, a person would have to wait for the script to complete before doing the next step. The server for interim storing of data between S3 and the database was a bottleneck because downloading just a few zip files at a time could fill it up. This previous process spanned several different technologies, storing the data temporarily in

several places, and employing several different style automation scripts (shell or Python), making it more complex and harder to maintain and de-bug.

New Data Pipeline

After looking at the entire process for ways to reduce waste and increase reliability, the new data pipeline has been redesigned **to reduce the long-term effort expended in upkeep.**



Data processing is now in the cloud (using Python programming and AWS services) to eliminate server space bottleneck. Data keeps itself up-to-date, report refresh time is seconds.

Table schemas were set up on top of the parsed data to enable SQL queries to only get specific data range needed.

AWS data is set to never expire and additional monthly cost is negligible

The data pipeline is now redefined. This process was created with the primary goal of running automatically in the background without needing human intervention. The new data pipeline starts from the step where the files are uploaded to S3.

The first new step is an Amazon Web Services Lambda function. This function named “test_parser” is a total rewrite of the original Python parser. Instead of downloading files, “test_parser” parses files and saves them back into AWS. Rather than one long script that was manually initiated like its predecessor, the parser code was grouped into classes following object-oriented programming best practices:

lambda_function.py – This script processes the triggers and handles all reading and writing to AWS. The trigger is either a file being uploaded to “ultra-mfg-prod” or a config file called "object_key_prefixes.json" being uploaded to the bucket "data-upload-testing.” The config file allows controlled parsing of historical files specified by the file prefix.

testmetrics.py – This python file defines a class TestMetrics. A TestMetrics instance is created for each JSON file found by the lambda_function. Each TestMetrics parses the JSON for information specific to the test type and metrics measured. TestMetrics is an improvement over the former parser as it applies data transformations to get the intended upper and lower metric bounds. Another improvement over its predecessor is that this parser maps any metric values embedded in lists into useable rows of data, and further maps string values to integer or float values when doing so would not cause an error. This saves a lot of time for analysts building dashboards from the data.

teststation.py -- This python file defines a class TestStation. This class parses a JSON file and the zip file name to gather information at the level of the testing session: date of the testing, serial number of the system the tests were run on, and serial numbers of the modules.

After parsing, the previous pipeline uploaded the data straight into a database without further processing. The database’s role then was solely as passive data storage.

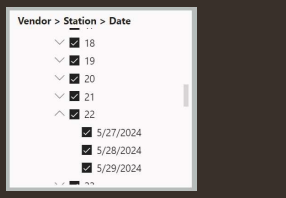
Identifying the lost opportunity to reduce data redundancy by utilizing a relational data schema as well as enabling massive scale, the new process deviates from the old process to take advantage of the AWS Athena service. As the data is parsed it is “normalized” to a certain extent -- “normalizing” used in this case in the sense of putting repeating data in a separate table and creating a relationship with foreign keys to the corresponding rows in related tables. The parser outputs data particular to the system hardware in one area of AWS S3 (*Testmetrics* bucket), and data particular to each metric values in another area of AWS S3 (*Teststations* bucket.) SQL queries were written and ran one-time in Athena to impose a schema ongoing on top of the data in these S3 buckets. Appendix 1 contains the database schema diagram and two reference SQL queries. A Glue Crawler service was not used to automatically discover schemas, but optionally could be turned on in future.

The last step of the new data pipeline process is getting the data into Power BI, the company standard dashboard software. Previously, the tables of data were manually exported into csv files and manually imported as csv files into dashboards. The new process is a one-time set up of a data connector to Athena tables.

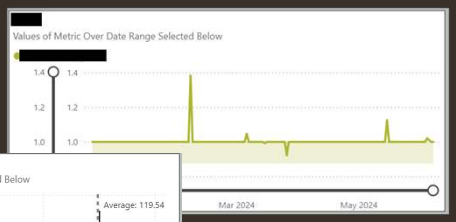
Dashboard

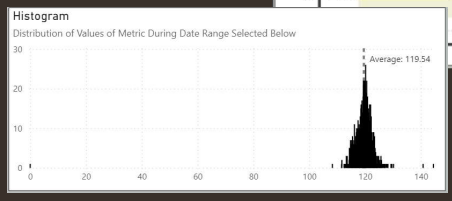
A new interactive visualization dashboard was iteratively designed to showcase insights from the parsed data in Power BI.

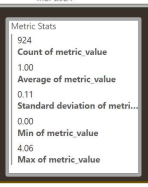
Slicer Options



Visualization Options



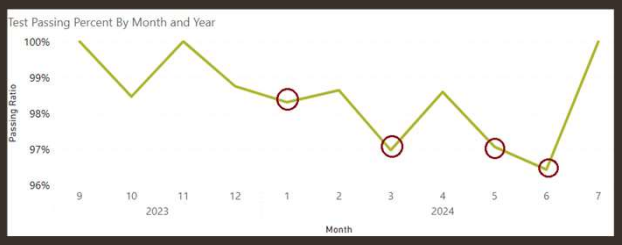




Change Detection Analysis

An additional question from the organization, “How can changes over time in the mean value of measurements be detected?,” led to **experiments in change detection**. These experiments were done with simulated data in Excel.

Exponential Weighted Moving Average (EWMA) Experiment



Detecting a trend of the individual measurements having increasingly larger deviations from the initial mean is recommended to be done with the Shewhart formula (as described at https://en.wikipedia.org/wiki/Shewhart_individuals_control_chart.) This formula is good for detecting “runs” of measurements when the mean of the distribution of values has changed.

The Shewhart formula starts with calculating the absolute value of the deviation of each individual

measurement from the measurement just prior to it. The control limit is calculated as the metric mean value from a period that was known to be in control plus 2.66 (a constant chosen based on statistical principles) times the mean absolute value of the deviations. An example worksheet Shewhart_example.xlsx is included with this report.

To check if the average rate of test passing per month is changing requires detecting a trend of much smaller deviations in the mean of measurements. This is recommended to be done with an Exponential Weighted Moving Average (EWMA) calculation. The EWMA formula was taken from NIST.gov website section titled "6.3.2.4.EWMA Control Charts." The EWMA calculation starts with multiplying the average for that month by a constant called alpha, typically 20%. That value is added to the previous period EWMA multiplied by (1 – alpha). This calculation is recursive, so it doesn't play nicely with Power BI, but can be easily done in Excel.

Here is a sample of EWMA analysis with theoretical data on the ratio of tests that passed measured every month.

Results

This revamped reporting system received positive feedback in the client survey and during the presentation to organization leadership.

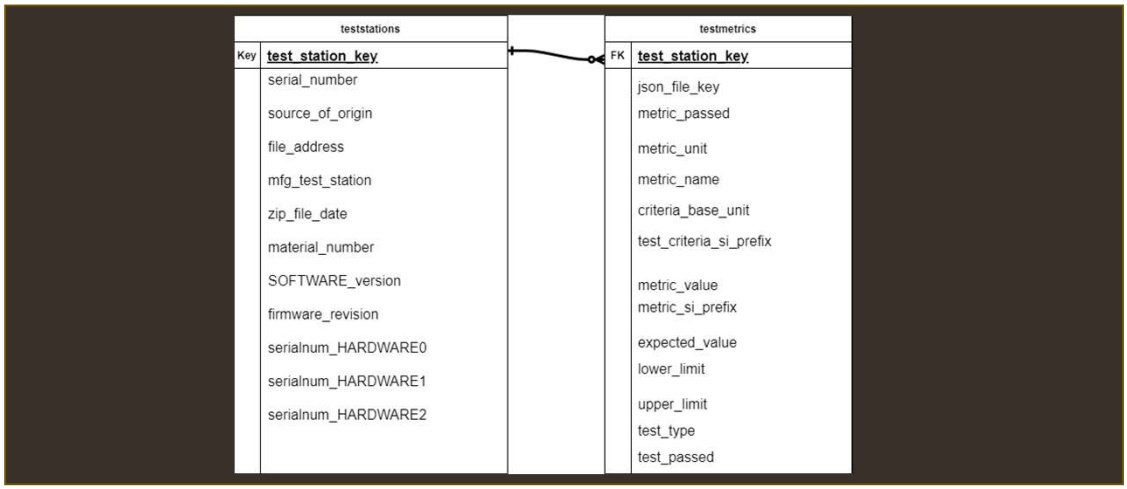
I also received personal satisfaction in crafting a solution to a business problem in an area that is new to me: Learning organization's goals, understanding current state, and designing a solution that targets value creation and elimination of waste

Thank you.

Appendix

Appendix 1

This is a reference diagram for demonstration of the table schemas in AWS Athena.



This is a reference diagram for demonstration of the table schemas in AWS Athena. There is a single teststations table. There are tables following the schema displayed here for “testmetrics,” each one of them corresponding to a unique combination of metric_unit test type, but all of those tables follow this reference schema. There can and often are more than one row having the same test_station_key, json_file_key, metric_name per metrics table, but they are only ever related back to one row with a unique test_station_key in “teststations.” The parts of the field names in

UPPERCASE are obscuring the specific hardware acronyms for confidentiality reasons.

Appendix 2

Here is a SQL written to impose the schemas on top of the data files.

```
CREATE EXTERNAL TABLE IF NOT EXISTS `ultrasound-test-analytics`.`test_type_metric_unit` (
  `test_type` string,
  `test_passed` boolean,
  `json_file_key` string,
  `test_station_key` string,
  `metric_passed` boolean,
  `metric_name` string,
  `metric_unit` string,
  `criteria_base_unit` string,
  `metric_si_prefix` string,
  `metric_value` float,
  `test_criteria_si_prefix` string,
  `upper_limit` float,
  `lower_limit` float,
  `expected_value` float
)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://testmetrics-prod/test-type/metric-unit/'
TBLPROPERTIES ('classification' = 'json');
```

Here are two reference examples of the SQL used to one-time impose the schemas on top of the S3 files. The parts of the field names in UPPERCASE are obscuring the specific hardware acronyms for confidentiality reasons. The location in red font is the S3 URI which was copied from AWS.