|  |  |
| --- | --- |
|  | |
| Data Analysis and Visualisation Project  Data Vis and Comms CA – Report | |
| Module code : B8IT107 | |
| Ciaran Finnegan  Student No : 10524150  29/03/2020 |  |
|  |  |

Table of Contents

1 Project Overview 3

1.1 High Level Description – Software Quality Assurance Dashboards 3

1.2 Environment Assumptions 4

1.3 Project Execution Instructions 4

1.4 Data Privacy Considerations 5

2 Part 1: Understanding The Business Drivers 6

2.1 Business Objective 6

2.2 Subject Area for Analysis 6

2.3 Primary Goals and Objectives for the Data Visualisations 7

2.4 Building Blocks for the Quality Assurance Dashboards 8

2.5 Key Stakeholders 11

3 Part 2: Data Preparation 12

3.1 Collating Excel Data 12

3.2 Preparing the Production Defect Data Extract 14

4 Part 3: Data Ingestion and Manipulation 15

4.1 Data Ingestion Using PowerBI – Query Builder 15

4.2 Data Manipulation in Power BI – Post Query Editor 18

5 Part 4 : Data Analysis / Visualisation 25

5.1 Dashboard One Design – Defect Analysis Dashboard 25

5.2 Dashboard Two Design – Defect Priority Dashboard 43

5.3 Dashboard Three Design – Test Automation Dashboard 56

6 Conclusions 71

6.1 Interpretations from the Quality Assurance Visualisations 71

6.2 Challenges in the Quality Assurance Data Visualisations 72

7 References 73

7.1 The Visualisation Theme 73

# Project Overview

## High Level Description – Software Quality Assurance Dashboards

This document covers the design, implementation and observations on all parts of the March 2020 CA for the Data Visualisation and Communications module (BTIT107).

This project is based on real world requirements and data to provide a graphical view on the effectiveness of the Quality Assurance (QA) process within the development and Production deployment lifecycle for my company.

The business domain for this project can be characterised as follows;

* This is a software engineering company that develops an enterprise level product for the Financial Services industry.
* Products are typically deployed on client site within their local Production environment (not cloud based).
* The client base is approximately 100+ customers. There is a wide geographical spread of clients.
* The ‘product’ is built up from multiple components, which are developed and tested across multiple regions.
* A new version of the product is released every six months, and deployments are facilitated through an external consultancy function. This business unit records all defects raised during the lifetime of the product into a third party tool called JIRA.

I have taken actual data on defects raised during product development and built dashboards that combine this data with Production raised defects. The intention is to create a more refined reporting process on QA effectiveness, both in general and broken down over time/location/product component.



## Environment Assumptions

Reporting dashboards were developed and presented in PowerBI.

A number of EXCEL worksheets contained the source information on defects raised during the Product development lifecycle. These had been built up over a period of 3+ years and were formatted into identical EXCEL worksheets.

The Product defect data was a .*csv* file of 2000+ records, extracted from the production defect tracking system, which is a third party tool called Atlassian JIRA.

The underlying data preparation and manipulation processes for the dashboard visualisations are described in Section 3 and Section 4 of this document.

The visual output of this reporting process is described in Section 5 of this document.

To summarise, the key software components used in development of this project were;

* PowerBI
* Microsoft Excel
* Notepad++ v7.5.8 (for occasional ad-hoc data views)

## Project Execution Instructions

Building the PowerBI dashboards was a process of repeated data manipulation and processing to provide the final dataset suitable for the visualisation charts I developed.

This resultant *.pbix* file has been included with the CA2 submission and can be loaded into PowerBI;

* ***DataVizComm\_CA\_DefectAnalysisDashboard FINAL cf v1-30 290320.pbix***

## Data Privacy Considerations

This project is built with real world data collated within my company over approximately the last three years.

I felt it was a suitable candidate for the Data Visualisation and Communication CA, but I also intend to put it to use in real life and improve the standard of metrics that are employed to show QA effectiveness.

Obviously when dealing with real data there is a need to be aware of company and regulatory data policies.

To address these issues, I took the following steps with the product defect data;

1. Client names and mnemonics are captured by the JIRA defect tracking system for defects in production. These have all been removed from the ***.csv*** JIRA extract.
2. Similarly, names of anyone adding or tracking a defect ‘ticket’ have been removed, regardless of whether the individuals are internal employees or customers.
3. The product version numbers have been re-sequenced to obscure the actual release descriptions.
4. Product components and office locations have been altered.

For internal office use in the near future I will revert some of the above data changes, made for the CA, so that I can employee the dashboard in a current work setting.

My company has an enterprise wide PowerBI licence and I will be using the online version for report generation and presentation.

# Part 1: Understanding The Business Drivers

## Business Objective

Testing is an essential part of the Software Development Lifecycle. However, how do you know if the investment in QA is being effective? How can you measure the impact that process changes are having over time?

In my company the move in the last five years to an AGILE framework for software product development has coincided with additional investment in formal test engineers and new test automation technology.

The entire concept of ‘Quality Assurance’ has taken on a ‘whole team’ methodology, not just measured by how effective the testers are at finding bugs but by how effective the coders are at removing errors before the tester or the client find them.

Are these changes having a positive impact on product quality, and by extension customer satisfaction? Can we demonstrate the effectiveness (or lack of success) of these changes in a manner that is easily understood by all stakeholders.

This is the objective of the dashboard visualisations in this project.

## Subject Area for Analysis

A brief review of ISTQB recommended practices (the commonly adopted standard for QA processes) suggests a range of metrics to use to measure QA effectiveness.

In this assignment I have chosen to focus on an analysis of*the ratio of defects found during the development phase versus the amount of defects subsequently uncovered by the client*.

This includes a profile of the severity of issues raised by clients and the impact of increased test automation by the QA team, during the development phase.

This is a sequential analysis of defect profiles for the last seven product releases delivered by the company

**Note:** Each company product releases contains similar functionality, but with regular incremental updates/refinements. Development lifecycles tend to be approximately the same duration, with a relatively static number of developers and testers.

This allows for the comparative analysis between each release as internal company QA processes gradually change over time.

## Primary Goals and Objectives for the Data Visualisations

The cost of addressing production defects on client site is considerably higher than during development so the majority of bugs should be captured prior to product release.

However, the priority of a defect is also a key characteristic of the metrics. With commercial software spending time to fix every cosmetic defect may delay a release to market, which could result in loss of market share. For non-critical software, a sensible balance should be established to determine that the bulk of post release defects, although undesirable, are not responsible for serious processing flaws in a client production environment.

My dashboards are a means to graphically analyse that the company is striking the right balance in terms of defect capture, across components and development locations.

## Building Blocks for the Quality Assurance Dashboards

In recent history, each product release made by my company has produced a set of QA metrics for the formal Release Note documentation.

This has included Excel spreadsheets with the following data;

* Release Number
* Release Date
* Defects Found per Component (a typical release comprises 8+ separate components, some of which are developed in different locations and integrated before release).

Another Excel spreadsheet is updated on a less frequent basis with the following data elements;

* Post Release Period 1
* Post Release Period 2

These date periods reflect two consecutive deployment windows for each release. A key business objective is to determine the frequency/volume of defects raised by clients. The purpose of these Release Periods is to;

1. Determine if an unexpectedly high volume of defects are found early in the first post release phase, possibly indicating that the QA process was ineffective at capturing defects easily spotted by clients. Experience has shown that the first four-six months after a product release is the key timeframe.
2. User the second post release phase to effectively establish a ‘cut-off’ point for client defects. Practical experience has shown that with a large client base multiple clients will start to report the same defects, therefore skewing the metrics. In practice I have found that limiting the second period to approximately 8 months covered the vast majority of clients taking new product versions, while avoiding duplication of issues being captured across the client base.

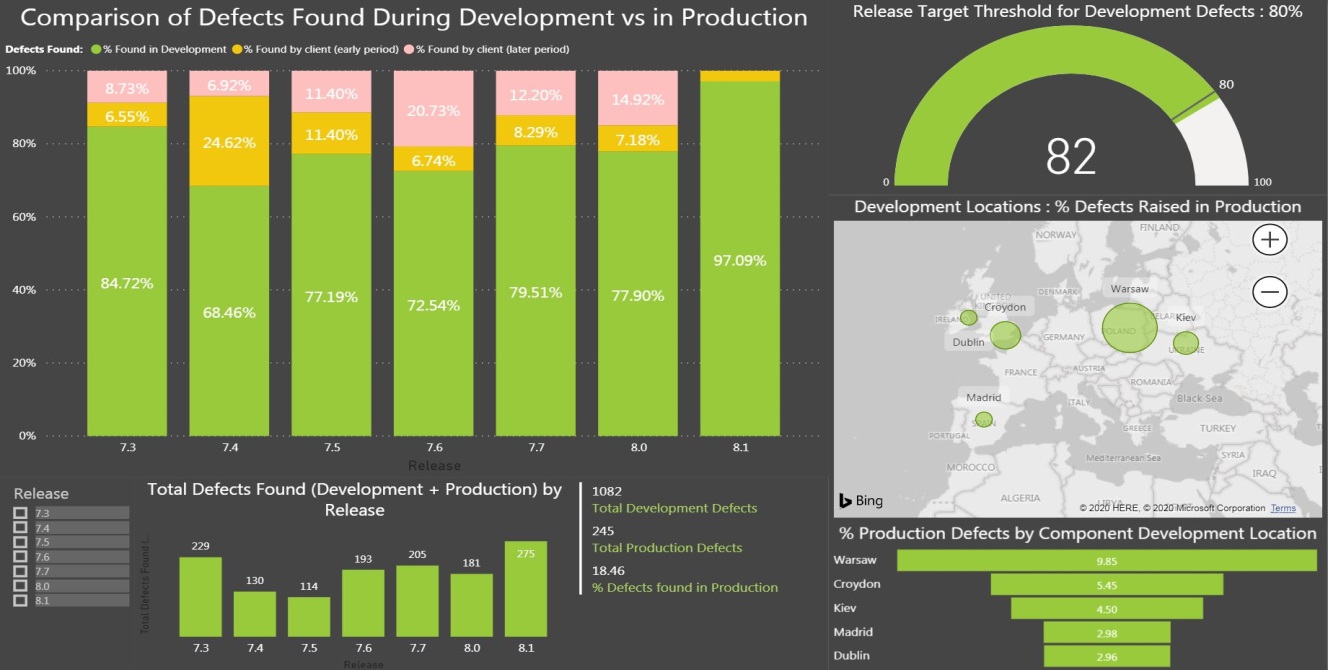
These dates are captured on a release by release basis. For the purposes of the CA I have provided the date periods but, for the sake of brevity, I have not elaborated on the historical reason for the date ranges themselves.

Previous internal processes involved laborious extraction of multiple spreadsheets, exported from the JIRA tool, to build up the production defect data.

Taking advantage of the functionality within PowerBI I have chosen to streamline the process by reading a full *.csv* extract, generated by JIRA, into PowerBI and build up a meaningful relationship between the data to produce the visualisations in this CA.

### Charting Development vs Production Metrics – What is the Trend?

Dashboard One focuses on defect volumes, and a graphical comparison of development numbers versus production.

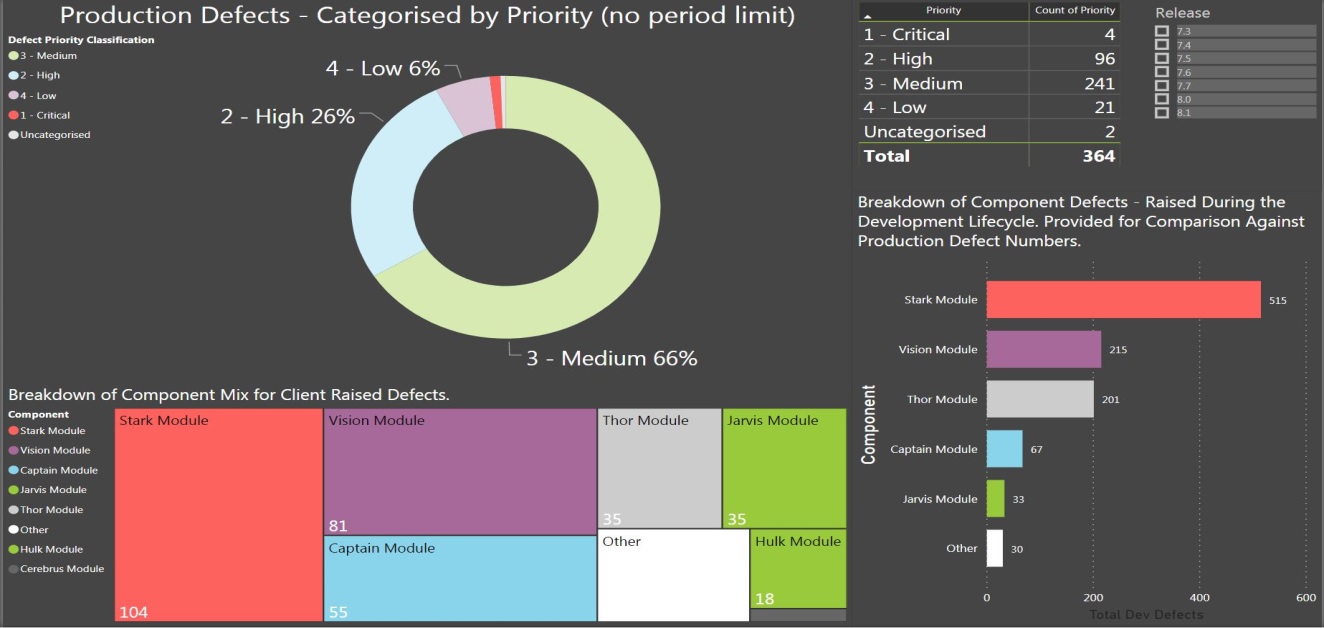


This project also aims to provide a graphical analysis of the ratio of defects based on component and location. Both of these metrics help identify a particular team unit responsible for a given segment of defects found in production.

This is not a ‘name and shame’ exercise, but rather an attempt to look at areas in the business that could possibly benefit from further investment in QA training or processes.

### How Serious are the Defects Raised in Production?

Dashboard Two provides additional information on the priority of the production defects, as assigned by external consultants and/or clients.



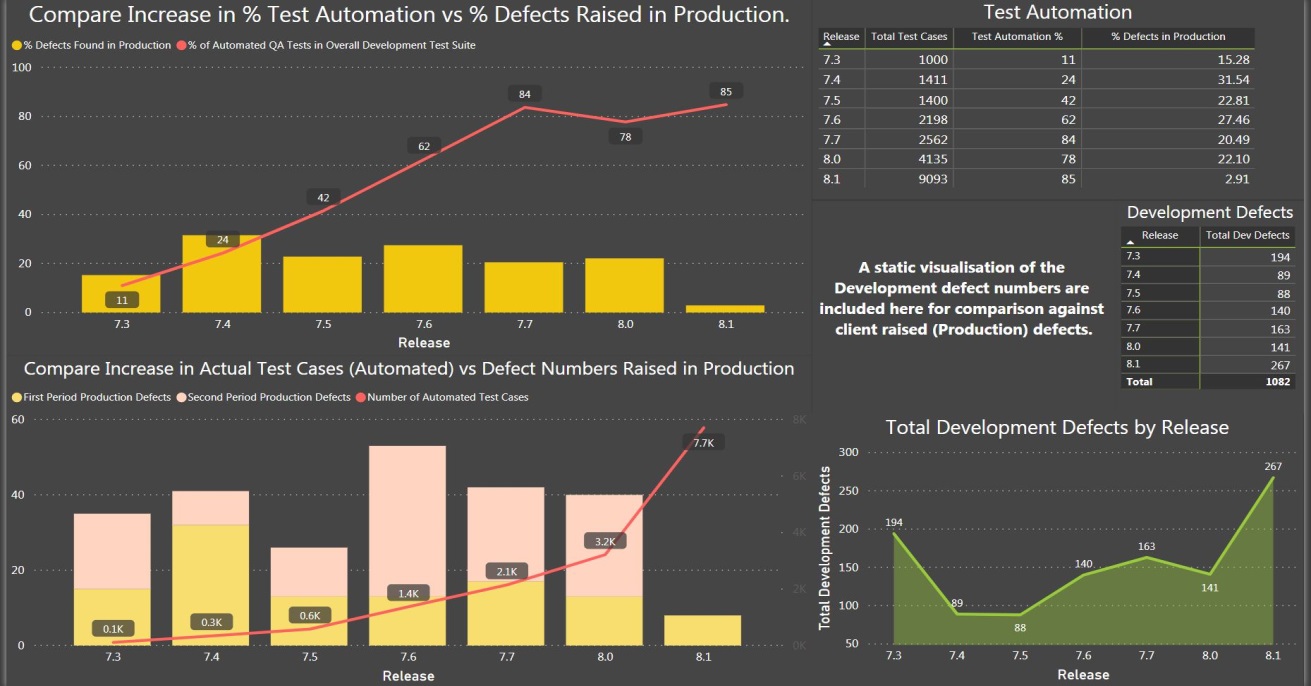
The dashboard aims to show that defects raised after a release has shipped tend to be in the lower priority range.

Defects in production are something to be avoided but are inevitable. The most realistic objective is to eliminate, as much as possible, product defects that cause major disruption on client site.

Clients tend to assign a slightly higher priority to defect tickets than those which would be assigned by a developer or QA engineer during development. Therefore defects should be at level three or lower when recorded on client site.

### Test Automation – Is It Working?

Dashboard Three contains visuals that show how the volume of test automation has increased over the period covered in the project. It overlaps this against the numbers of defects raised in both development and production.



Test automation is an integral part of any modern software development and test process. However, it can be expensive and challenging to deploy and business stakeholders will expect to see a return on this type of investment.

Dashboard Three has included a visual to make the case for the ongoing benefits of test automation.

(In this CA the test automation is Selenium based UI and API level tests to replicate/replace the traditional manual test process).

## Key Stakeholders

The reports produced by the work in this CA will ultimately lead to a more sophisticated and distributable reporting model for QA activity in my company.

The key stakeholders for the type of reporting produced in this project are therefore;

* **Senior Management.** Those that ask the question; can you prove that your QA processes deliver value?
* **Mid-level Management.** The people like me that want to see first-hand evidence that process changes we championed within our teams are actually working. Do we have an impressive way to convey this to higher levels of management?
* **Partners.** Those that deploy our software at client site need assurances that we are working to be proactive and continually trying to innovate with product quality.
* **Employees.** If team members are part of an effective delivery group, then can we measure this and reward appropriately?

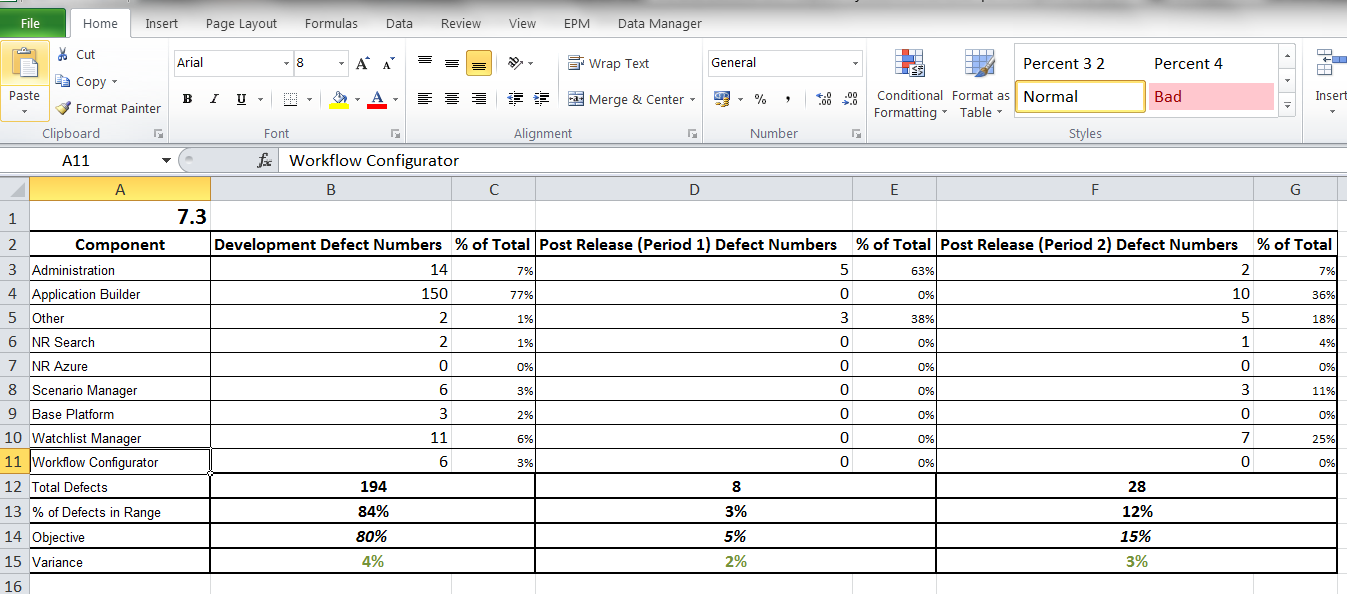
# Part 2: Data Preparation

## Collating Excel Data

Every time the software development test phase is completed for a company product, a list is manually prepared of the defects recorded.

This defect list is broken down by numbers of bugs per component.

The EXCEL sheet typically looks like this;

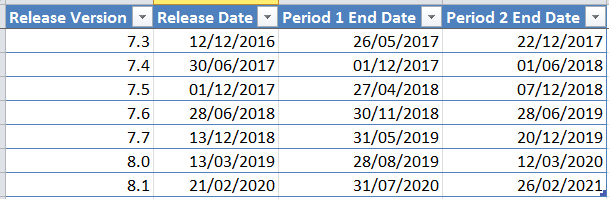


Seven of these sheets, one for each release in the project period, have been collated for the dashboard visualisation project.

PowerBI will reformat and extend the data into a structure that can be visualised in charts on the CA project dashboard (as described in Section 4 of this document).

In addition to this defect data from the release development phase, there are three other spreadsheets containing supplementary reference data that is essential for meaningful visuals;

* Release Date Information

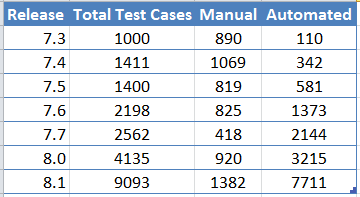


* Component Development Locations



The mapping of component names is used to reduce the numbers of elements on some of the visuals.

* Test Automation Numbers

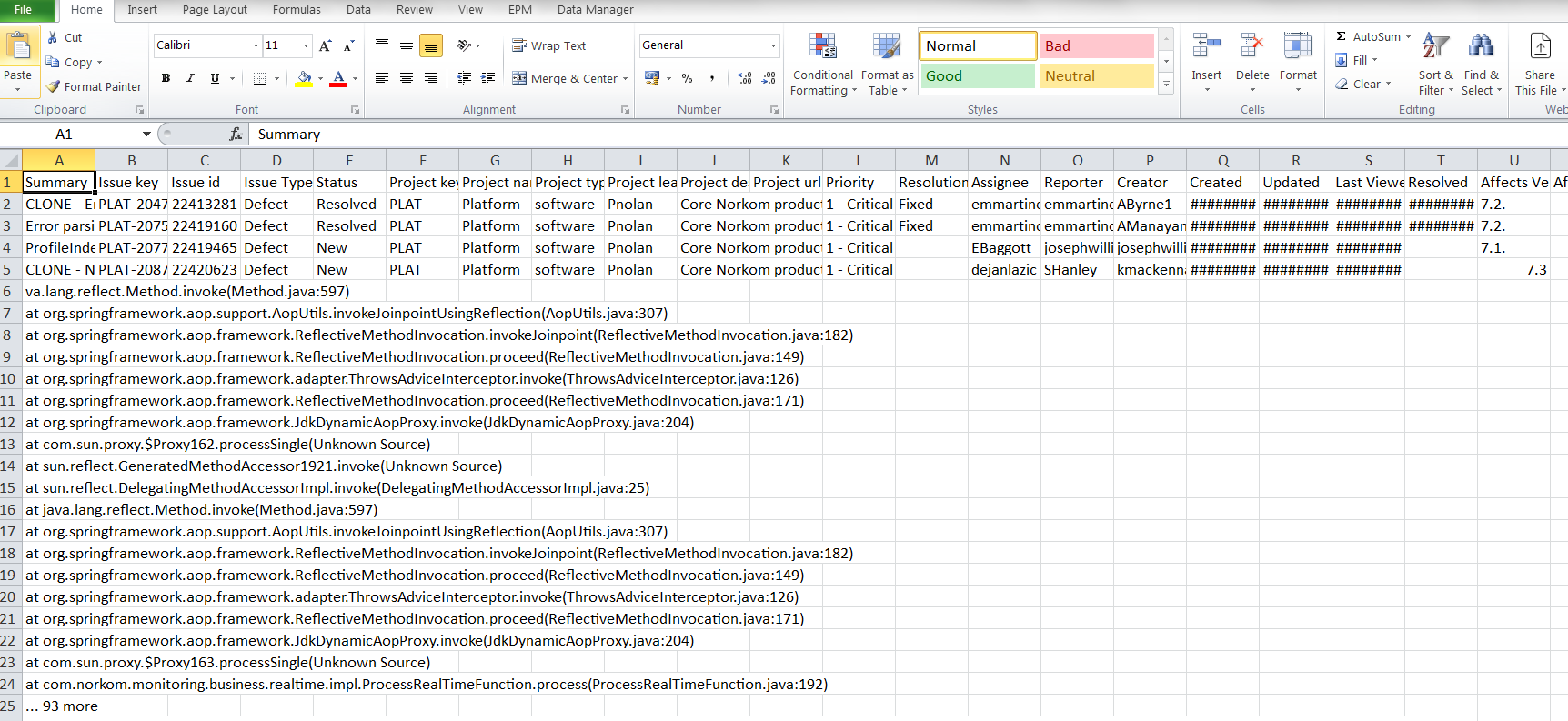


## Preparing the Production Defect Data Extract

The core of much of the dashboard data is a live extract of production defect data from the Atlassian JIRA tool, as provided by the company for use by consultants and clients.

The tool provides custom dashboards to allow me to generate a csv extract of production data at any given time.

The raw file looks like this;



The file approximately 2000 records in length with 30+ columns, which can be adjusted dynamically for each extract.

PowerBI is used to ingest this data and then reformat and extend as described in Section 4.

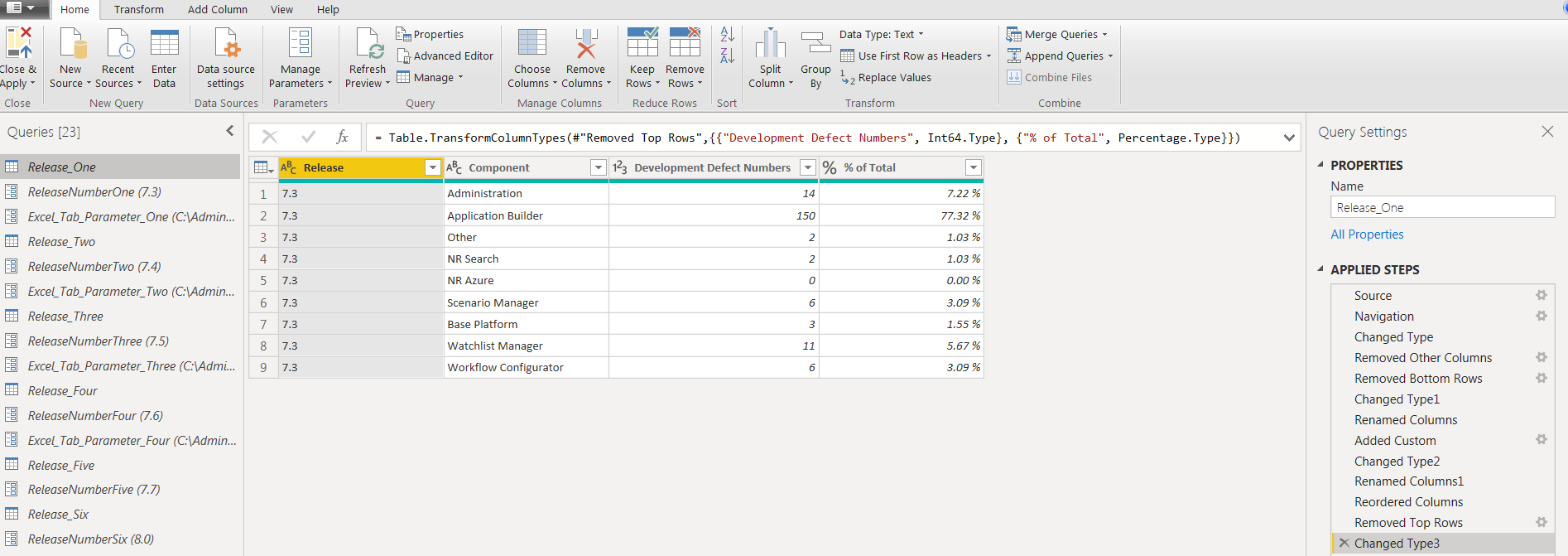
# Part 3: Data Ingestion and Manipulation

## Data Ingestion Using PowerBI – Query Builder

Before the construction of visuals within the dashboard there is a need to reformat and extend the raw data into a usable format.

### Development Defect Data

The screenshot below shows the end result in PowerBI of the Query Builder manipulation of the first of the release spreadsheets that capture development defects.

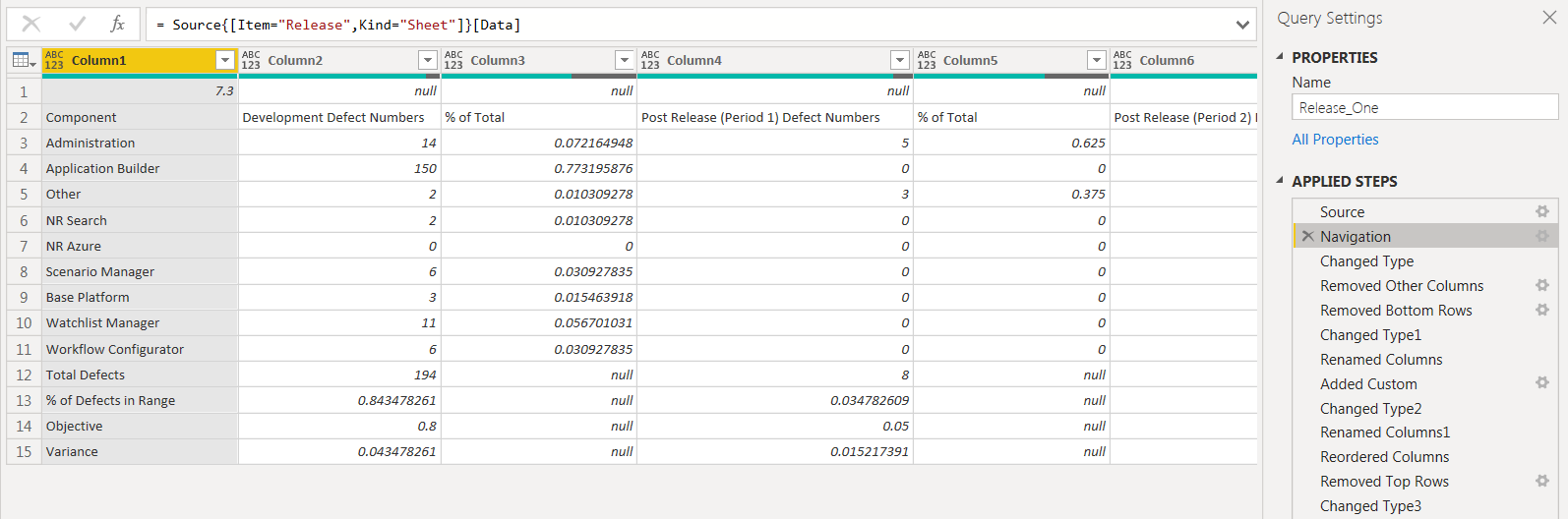


This shows the transformation of the data, as shown originally at the start of Section 3.1 of this document.

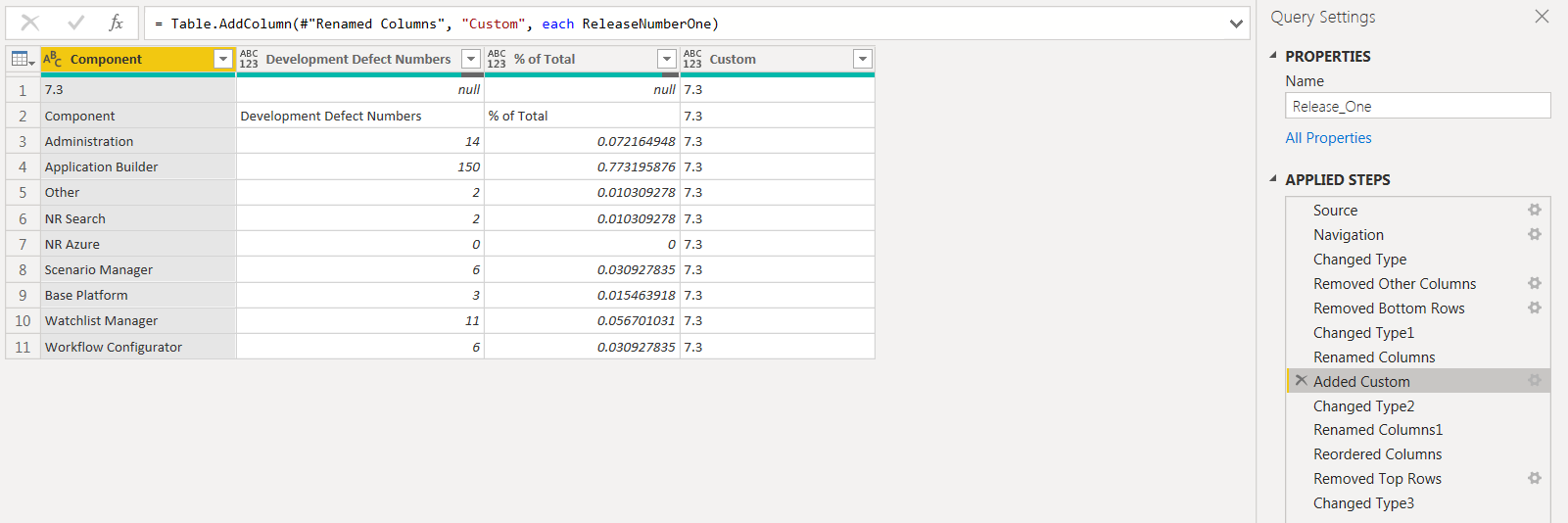
There are thirteen ‘Applied Steps’, as can be seen on the right.

Selecting some of the key steps we can see changes such as;

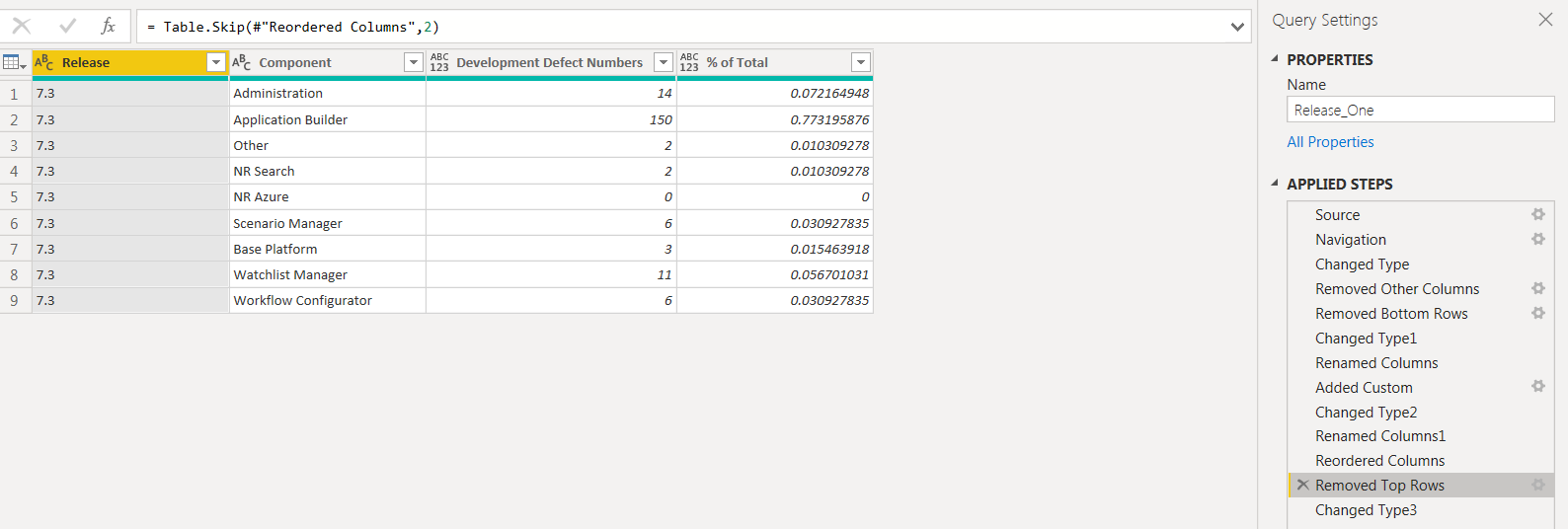
**Initial Navigation..**



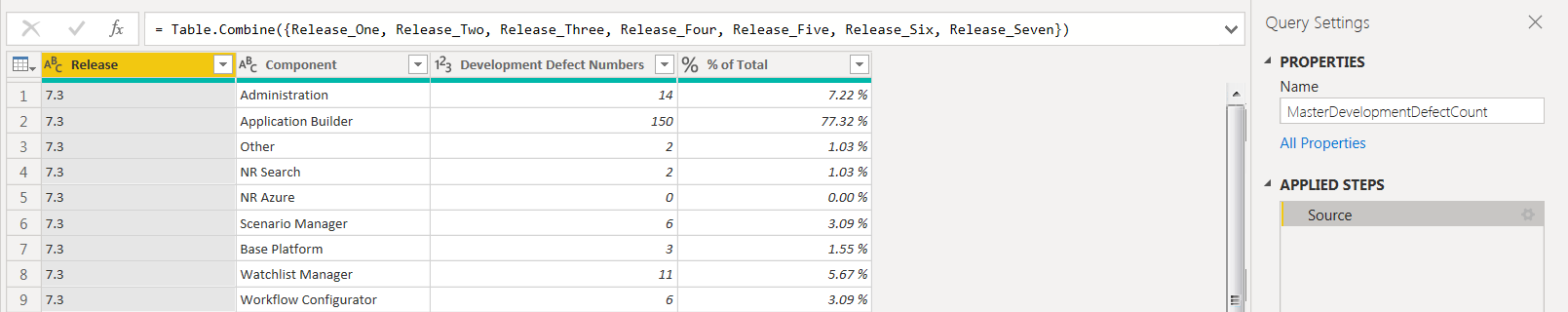
**Adding a new column to identify the release number..**



**Removing top rows..**



The last step in Query Builder combines all the individual release information into a single data source called : *‘MasterDevelopmentDefectCount’*



Collating the information into a single data source makes the building of the queries, which will be filtered by ‘Release’, much more straightforward.

The other development data tables undergo a similar transformation in structure but no additional columns are added in Query Builder.

### Production Data Extract

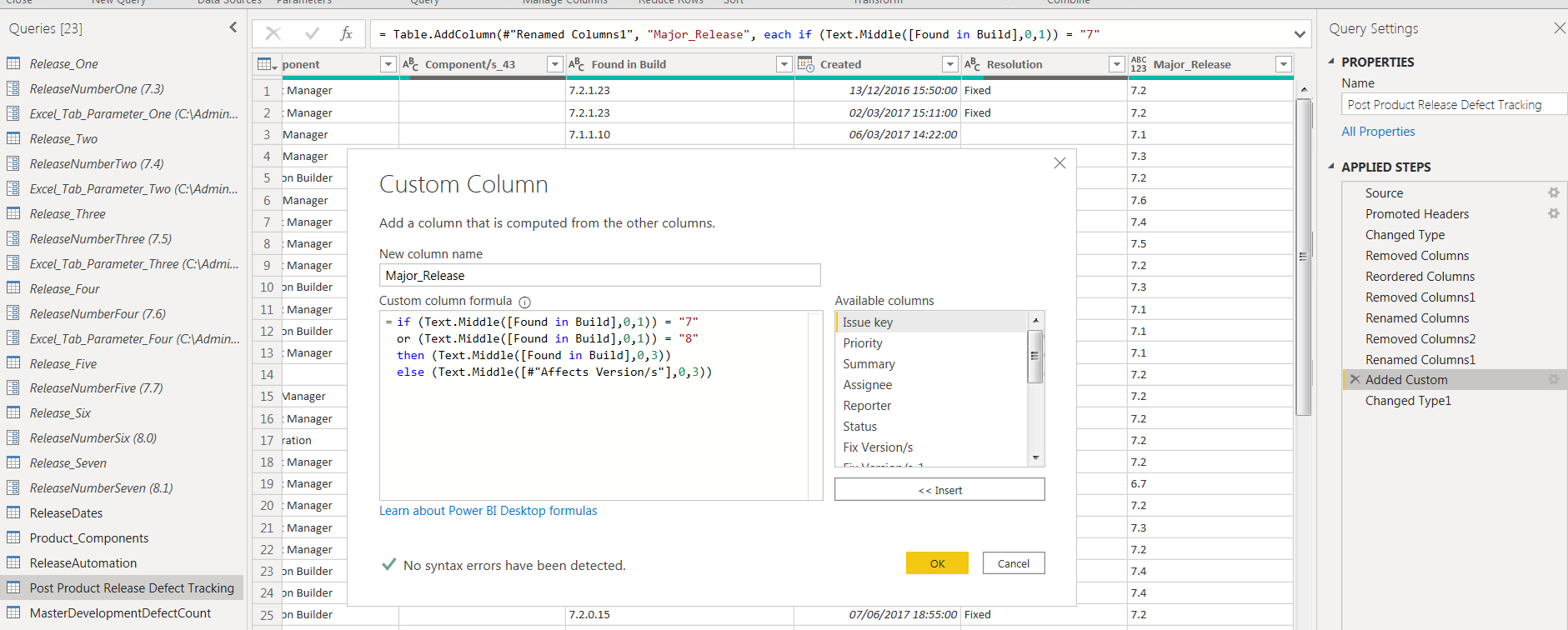
The .*csv* file from JIRA is ingested into PowerBI Query Builder and this contains the production defect data.

This is a much larger file than the development defect data.

The process of capturing defects in production uses a field called ‘*Found In Build*’ and this typically contains a description of the release that has too much version information. This information needs to be pruned so that we strip away any characters that are unnecessary.

For example, a build numbered ‘7.3.1.1’ in production needs to be converted to a simpler ‘7.3’ reference for the purpose of our dashboard visualisations.

A new column named ‘Major\_Release’ is added to the data source based on the custom operations displayed in the screenshot below.



This transformation is important to allow the creation of relationships between production and development data sources in PowerBI, which will drive dynamic filtering in the dashboards.

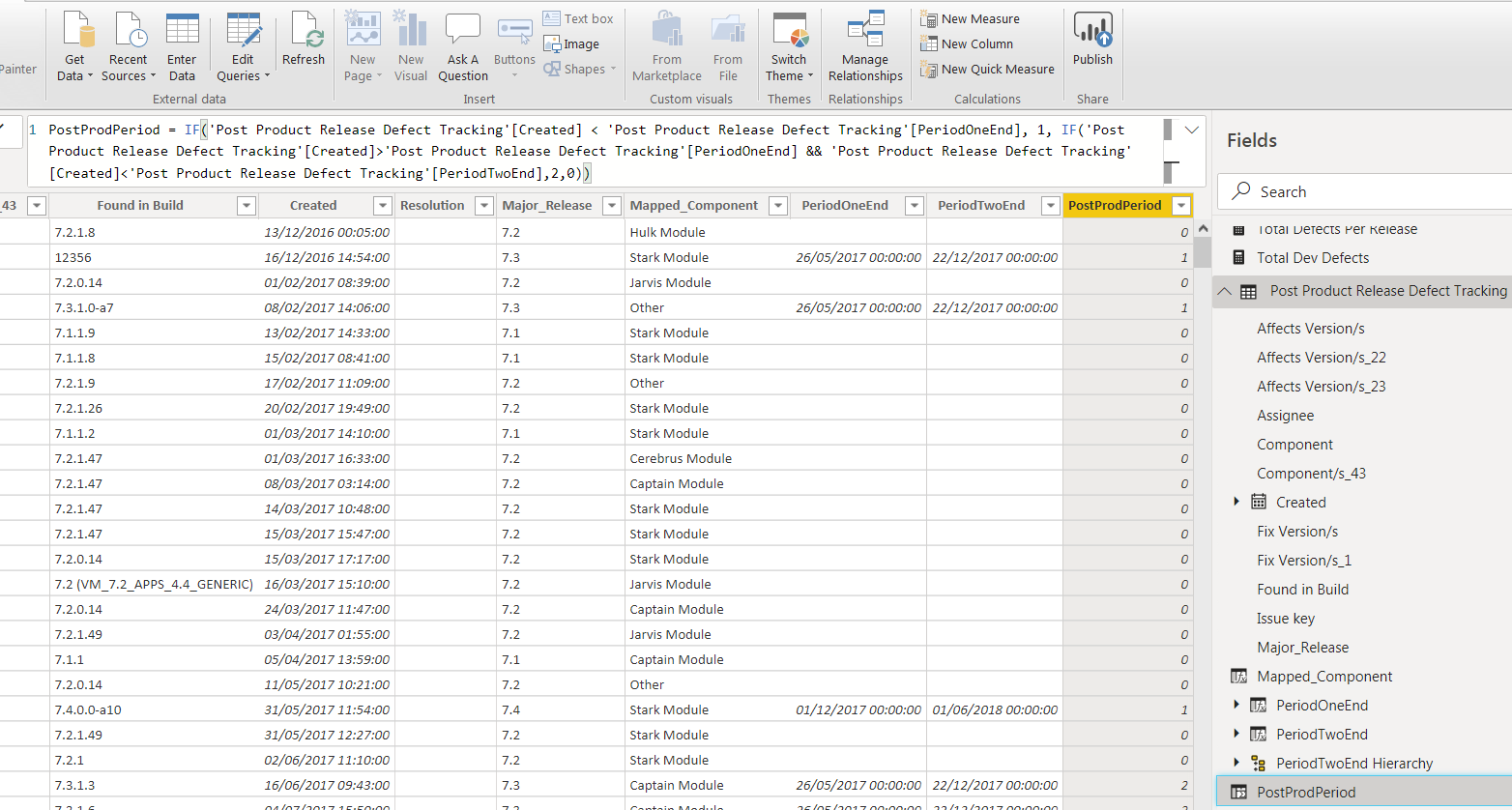
## Data Manipulation in Power BI – Post Query Editor

Once the data operations were finished in Query Builder and I selected the ‘*Close and Apply*’ option, it was necessary to work on creating new ‘Measures’ and adding columns.

The addition of these fields allows for much greater richness of detail in the resultant visualisation in the dashboards.

### New Columns

PowerBI provides a powerful mechanism to add new columns to the data sources (as can be seen in the screenshot below). This extends existing data and enables more interesting visualisations.



#### New Column : Mapped\_Component

Data Source : ***MasterDevelopmentDefectCount***

Logic :

Mapped\_Component = LOOKUPVALUE(Product\_Components[Mapped Component],Product\_Components[Component],MasterDevelopmentDefectCount[Component])

Purpose : The production defect data is somewhat untidy with component descriptions so both the development and production data sources are being filtered/changed based on the Product Component table.

This new column contains the adjusted component description for the development defect data source.

#### New Column: Mapped\_Component

Data Source : ***Post Product Defect Release Tracking***

Logic :

Mapped\_Component = LOOKUPVALUE(Product\_Components[Mapped Component],Product\_Components[Component],'Post Product Release Defect Tracking'[Component])

Purpose : The production defect data is somewhat untidy with component descriptions so both the development and production data sources are being filtered/changed based on the Product Component table.

This new column contains the adjusted component description for the production defect data source.

#### New Column : PeriodOneEnd

Data Source : ***Post Product Defect Release Tracking***

Logic :

PeriodOneEnd = LOOKUPVALUE(ReleaseDates[Period 1 End Date],ReleaseDates[Release],'Post Product Release Defect Tracking'[Major\_Release])

Purpose : The production defect data is broken into three intervals but only the first two are captured for the dashboard visuals. This column reads the **first** period end date from the release date reference data source and is used in the logic elsewhere in a new column for the *‘Post Product Defect Release Tracking’* data source.

#### New Column : PeriodTwoEnd

Data Source : ***Post Product Defect Release Tracking***

Logic :

PeriodTwoEnd = LOOKUPVALUE(ReleaseDates[Period 1 End Date],ReleaseDates[Release],'Post Product Release Defect Tracking'[Major\_Release])

Purpose : The production defect data is broken into three intervals but only the first two are captured for the dashboard visuals. This column reads the **second** period end date from the release date reference data source and is used in the logic elsewhere in a new column for the *‘Post Product Defect Release Tracking’* data source.

#### New Column : PostProdPeriod

Data Source : ***Post Product Defect Release Tracking***

Logic :

PostProdPeriod =

IF('Post Product Release Defect Tracking'[Created] < 'Post Product Release Defect Tracking'[PeriodOneEnd], 1,

IF('Post Product Release Defect Tracking'[Created]>'Post Product Release Defect Tracking'[PeriodOneEnd] && 'Post Product Release Defect Tracking'[Created]<'Post Product Release Defect Tracking'[PeriodTwoEnd],2,0))

Purpose : Using the new columns described above, a flag is set to indicate of the defect row is in Period 1 or Period 2 for a release. (A zero reference indicates that the defect falls beyond the timeframe for tracking production issues).

#### New Column : Test Automation %

Data Source : ***ReleaseAutomation***

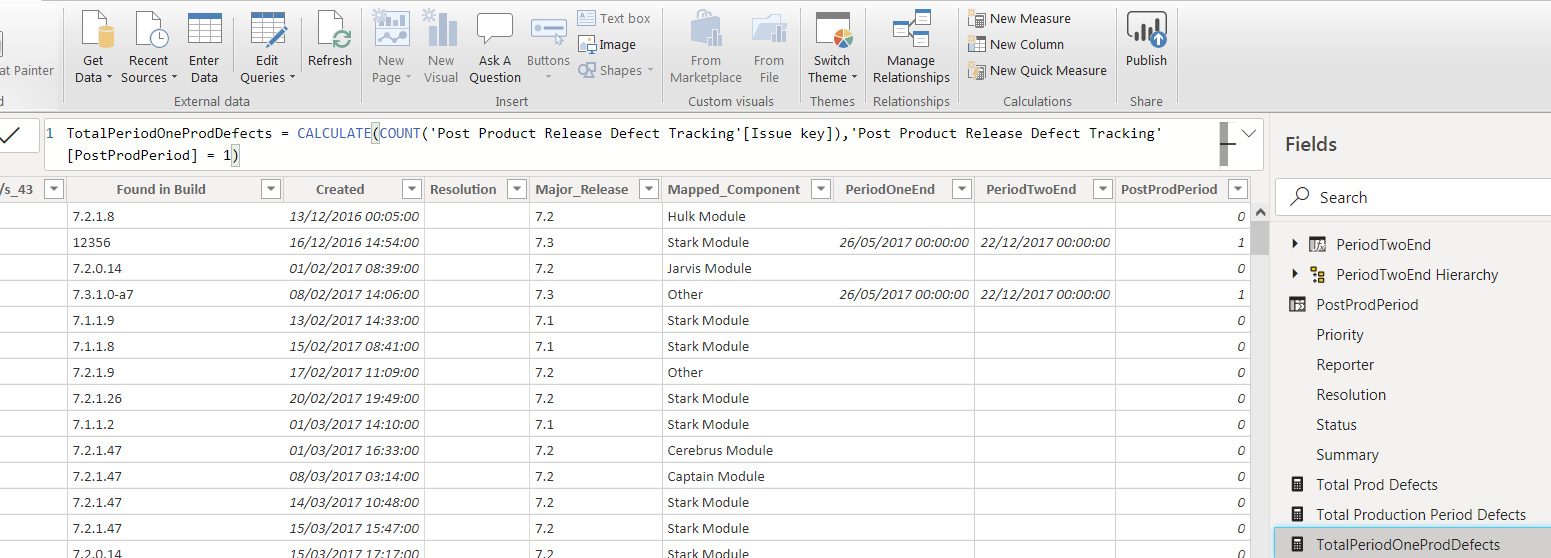
Logic :

Test Automation % = (ReleaseAutomation[Automated]/ReleaseAutomation[Total Test Cases]) \* 100

Purpose : Used in tabular and bar chart data to convert automated test case numbers into percentage values, which are then shown in dashboard visualisations.

### New Measures

‘Measures’ do not add extra columns to the data sources in PowerBI, but they do allow for additional variables to be created to enhance dashboard visualisations. An example is provided in the screenshot below.



The major PowerBI ‘Measures’ created for the CA dashboards are;

#### Measure : Total Dev Defects

Logic : Total Dev Defects = SUM(MasterDevelopmentDefectCount[Development Defect Numbers])

Purpose : The number of defects recorded during the development phase of each release.

#### Measure : TotalPeriodOneProdDefects

Logic : TotalPeriodOneProdDefects = CALCULATE(COUNT('Post Product Release Defect Tracking'[Issue key]),'Post Product Release Defect Tracking'[PostProdPeriod] = 1)

Purpose : The count of defects recorded in production by clients during the ‘Period 1’ phase for a given product release.

#### Measure : TotalPeriodTwoProdDefects

Logic : TotalPeriodTwoProdDefects = CALCULATE(COUNT('Post Product Release Defect Tracking'[Issue key]),'Post Product Release Defect Tracking'[PostProdPeriod] = 2)

Purpose : The count of defects recorded in production by clients during the ‘Period 2’ phase for a given product release.

#### Measure: Total Production Period Defects

Logic : Total Production Period Defects = [TotalPeriodOneProdDefects] + [TotalPeriodTwoProdDefects]

Purpose : The count of defects recorded in production by clients during the entire tracking phase for a given product release (but not those defects added for a release after the end of the tracking period).

#### Measure : Total Defects Per Release

Logic : Total Defects Per Release = [Total Dev Defects] + [TotalPeriodOneProdDefects] + [TotalPeriodTwoProdDefects]

Purpose : Combination of all defects ever recorded for a release (except those falling outside of the production tracking period).

#### Measure : % Dev Defects

Logic : % Dev Defects = ([Total Dev Defects]/[Total Defects Per Release]) \* 100

Purpose: Proportion of defects found during development as a percentage of the overall number recorded for a release.

#### Measure : % Period 1 + 2 Prod Defects

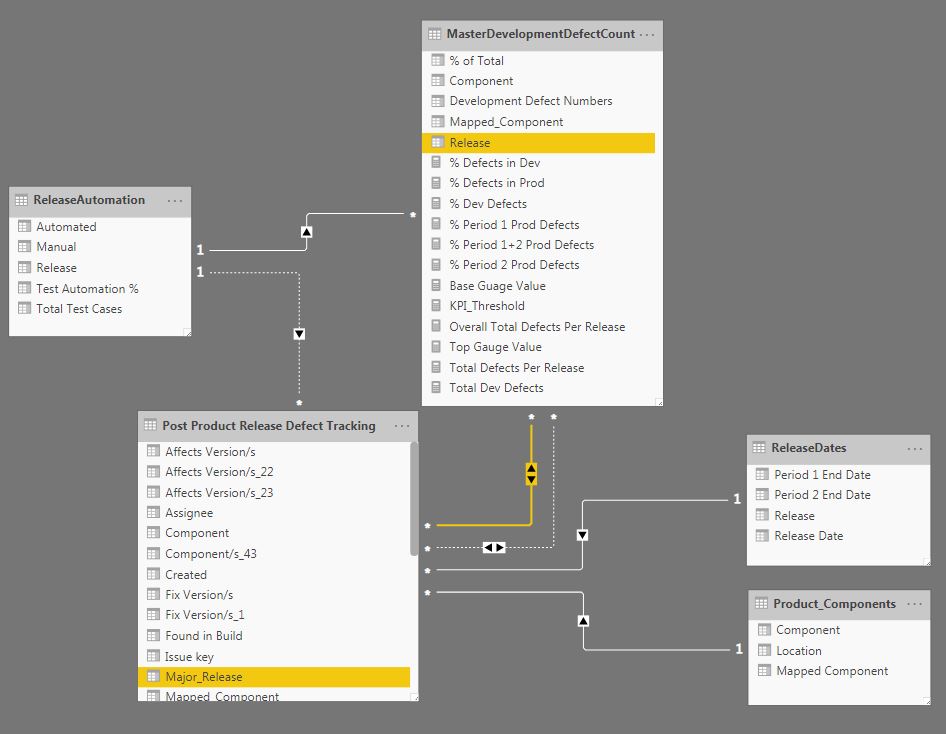
Logic : % Period 1+2 Prod Defects = [% Period 1 Prod Defects] +[% Period 2 Prod Defects]

Purpose : Proportion of defects found during production deployment period as a percentage of the overall number recorded for a release.

### Data Source Relationships

To make the dashboards dynamic and filter between release versions, it is necessary to add relationships between the data sources.

The diagram below displays the connections established between the PowerBI data sources;



The relationship between ‘Release’ in the *‘MasterDevelopmentDefectCount’* data source and the *‘Post Product Release Defect Tracking’* is highlighted in the above diagram.

This is a key connector linking the defect tracked per release in production to the defect profile (per release) during development.

Other key data source relationships are;

**Release**

*ReleaseAutomatio*n <-> *Post Product Release Defect Tracking*

*ReleaseAutomatio*n <-> *MasterDevelopmentDefectCount*

*ReleaseDates* <-> *Post Product Release Defect Tracking*

*Post Product Release Defect Tracking* <-> *MasterDevelopmentDefectCount*

**Component**

*Product\_Component* <-> *Post Product Release Defect Tracking*

*Post Product Release Defect Tracking* <-> *MasterDevelopmentDefectCount*

The cardinality for these relationships can be seen in the 1:\* style references between the data sources in the above screenshot in Section 4.2.3.

# Part 4 : Data Analysis / Visualisation

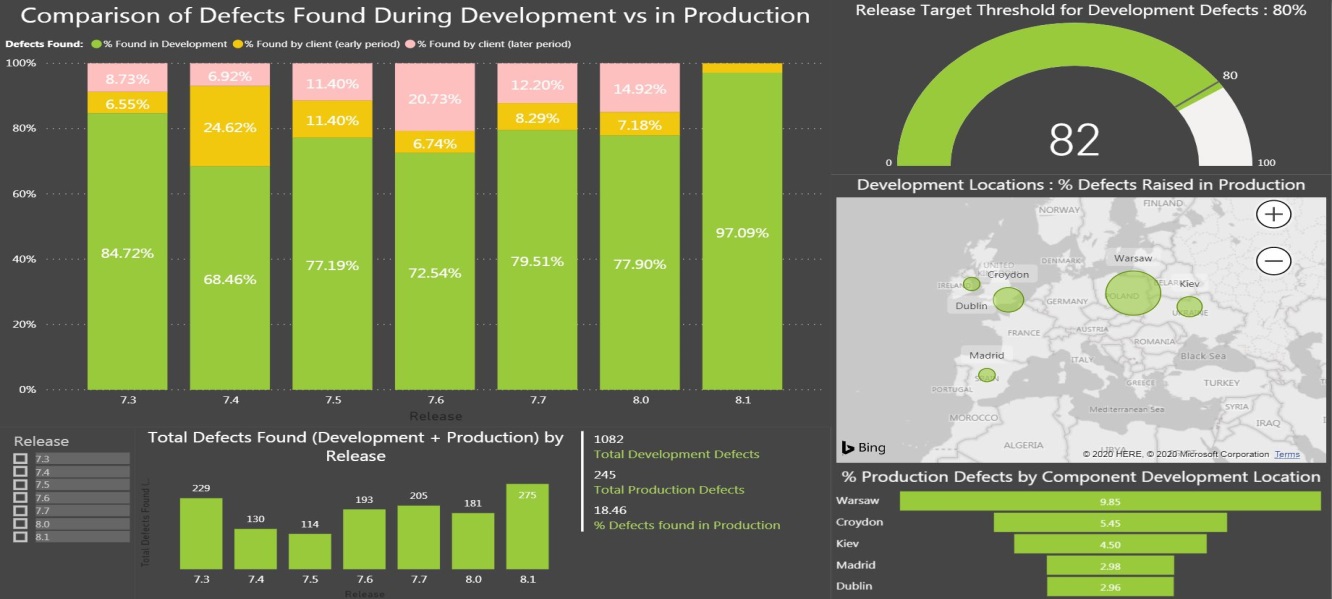
## Dashboard One Design – Defect Analysis Dashboard

The purpose of this dashboard is to show the ratio of software defects found during the development phase, versus those found in production environments by clients/external consultants.

**Dashboard 1: Defect analysis Dashboard**

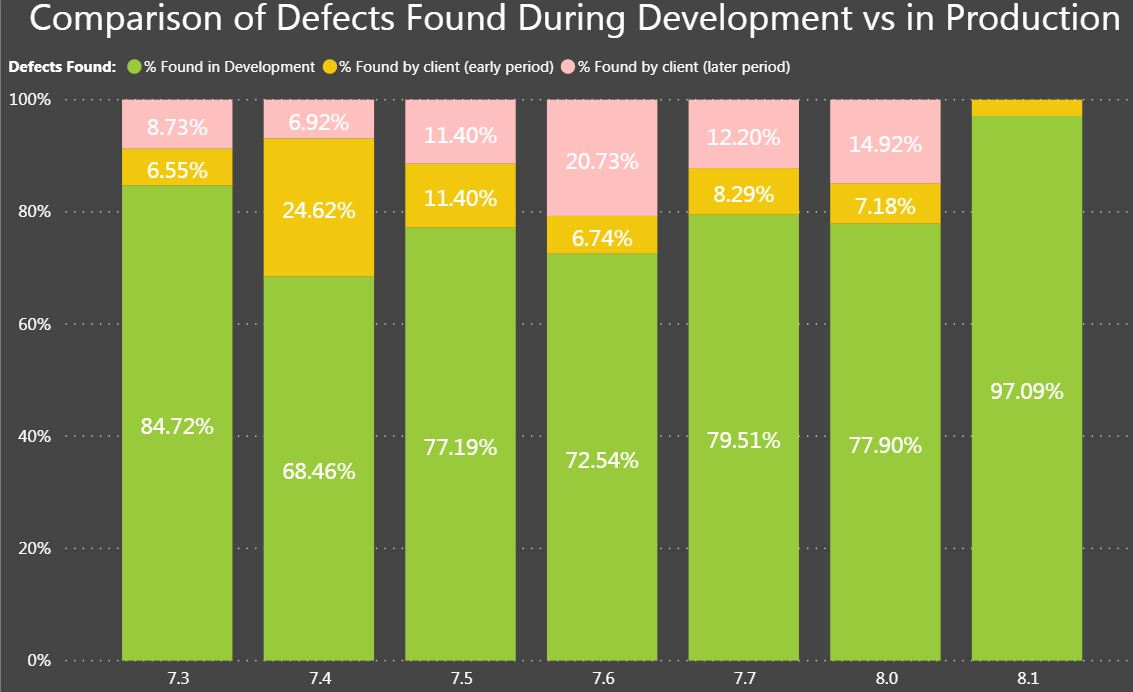
|  |  |
| --- | --- |
| **Dashboard Component** | **Visualisation (reading clockwise from top)** |
| **Tile 1** | **Comparison of Development vs Production Defects** |
| **Tile 2** | **Release Target Thresholds.** |
| **Tile 3** | **% Production Defects by Development Location (Map)** |
| **Tile 4** | **% Production Defects by Development Locations (Funnel)** |
| **Tile 5** | **High Level Metrics on Development Defects vs Production** |
| **Tile 6** | **Total Defects Found per Release** |

The first dashboard is a collection of visuals intended to show how effective the company QA process has been, over the period of the most recent product releases, in detecting defects before being shipped to clients.



### Tile 1.1: Comparison of % Defects Found in Development vs Production

This tile is a 100% stacked bar chart to show the ratio, in percentages, of the volume of defects found during the software development lifecycle of recent releases, versus those found in production on client site.



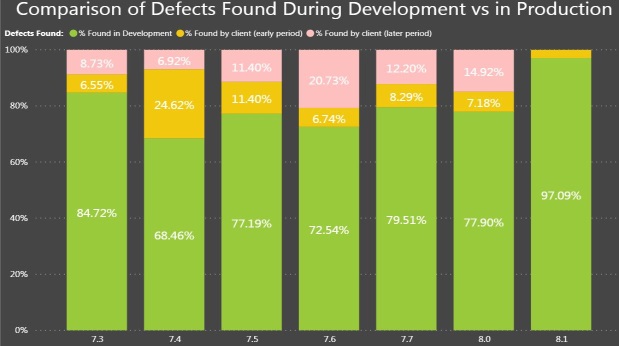
#### Purpose of Visualisation

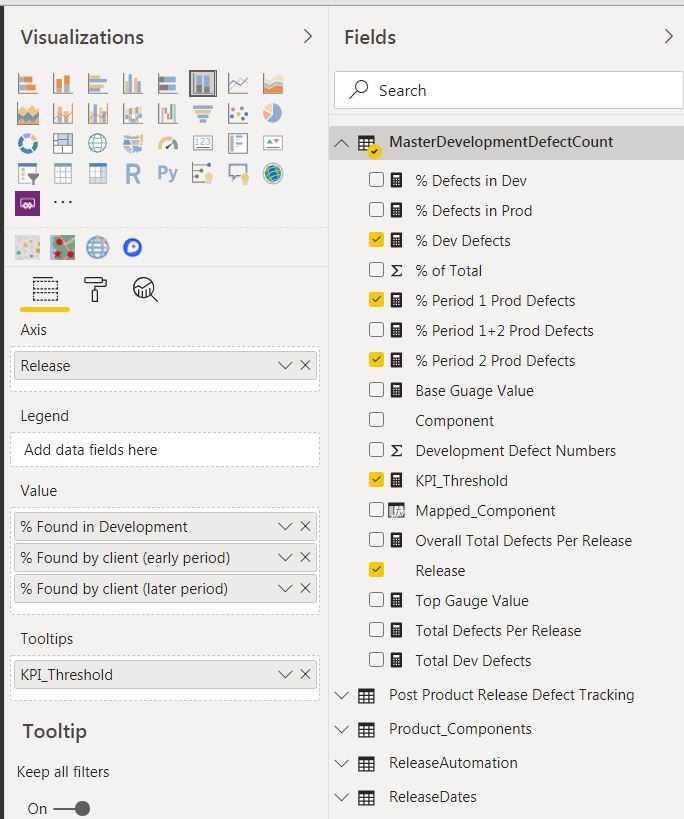
This tile provides a visual representation of how many defects are being caught by the company QA process during product development.

#### Visualisation Process

This graph is built solely from the *‘MasterDevelopmentDefectCount’* data source. The import of this data is described in more detail in Section 3 and Section 4 of this document.

Section 4 also describes the individual measures created for this dashboard in more detail.





The ‘Release’ axis is a field read directly from the Excel file into the data source but the values for the percentage of the development and production defects are calculated measures.

* % Dev Defects – this generates the initial ‘green’ bar.
* % Period 1 Prod Defects – this generates the ‘amber’ stacked bar.
* % Period 2 Prod Defects – this generates the ‘pink’ stacked bar.

A tooltip is provided for a hover-over, to advise the audience of the dashboard that 80% is the ideal minimum threshold for the proportion of development defects.

#### Why This Type of Visualisation?

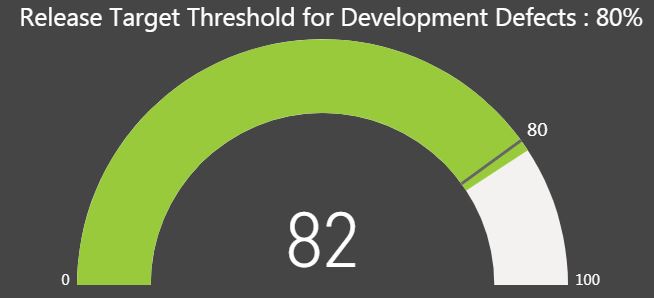
The 100% Bar Chart is used because we are looking at the entirety of defects in the development and production periods and comparing relative proportions.

#### Relation To Other Tiles in Dashboard.

Although a Slider control is provided on the dashboard, a user can also click on a given release bar in the visualisation and the other tiles will filter on that given release.

### Tile 1.2: Release Target Thresholds

This tile is a relatively simple gauge visual used to emphasise when the development defect detection rate has (or has not) achieved the ideal minimum threshold – 80% of defects captured during the development process.



#### Purpose of Visualisation

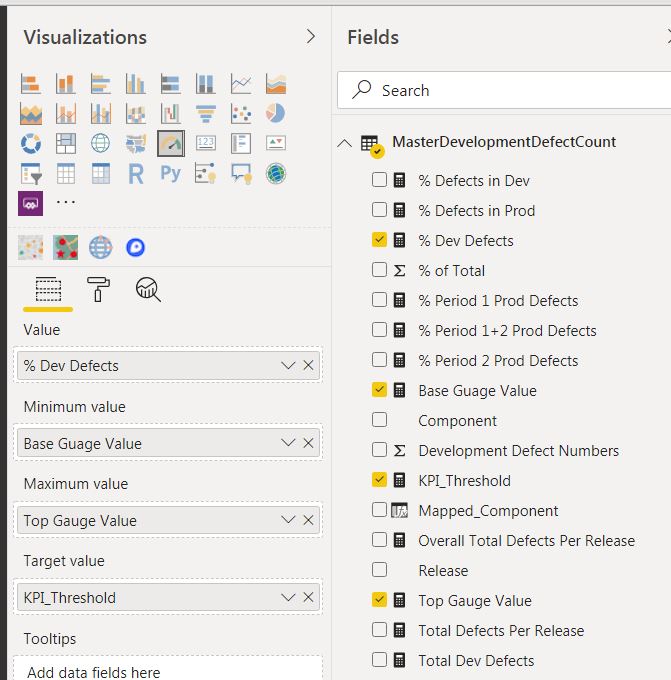
The user of the dashboard will initially see a default gauge value that combines metrics for all releases, showing how well the QA process is working at capturing bugs during the development phase.

The visualisation will change focus on specific product release(s) based on user selection filters elsewhere on the dashboard.

#### Visualisation Process

As with Tile 1 only the *‘MasterDevelopmentDefectCount’* data source is used and the visualisation is really driven by just the measure ‘% Dev Defects’





The upper and lower bound values of the gauge are hard coded to reflect a 0 – 100% range, as is the KPI target of 80%.

#### Why This Type of Visualisation?

This is a simple gauge to show if the development QA process is hitting the desired target for defect detection.

It can focus on one or more releases or all releases in the project period.

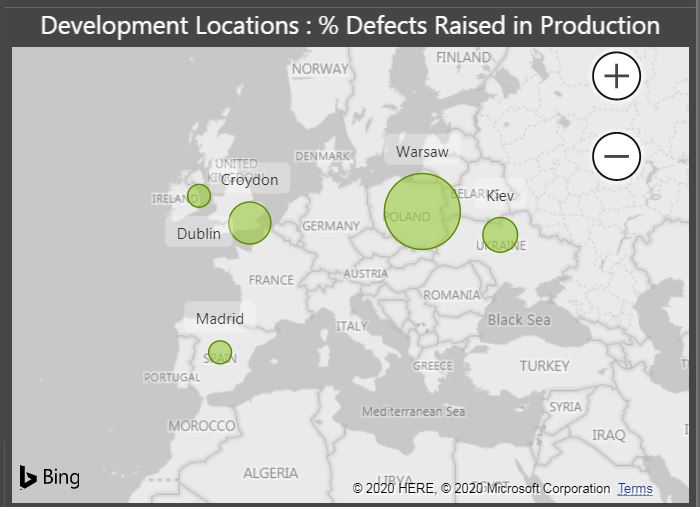
#### Relation To Other Tiles in Dashboard.

The KPI target value is static but the value for the actual percentage gauge will alter depending on the ‘Release’ selected in the other bar charts on the dashboard.

### Tile 1.3: % of Production Defects by Development Locations

This tile shows a map of company offices where software development and QA activity takes place.

The ‘bubble’ sizes reflect a relative proportion of defects found in production for components developed in these locations.



#### Purpose of Visualisation

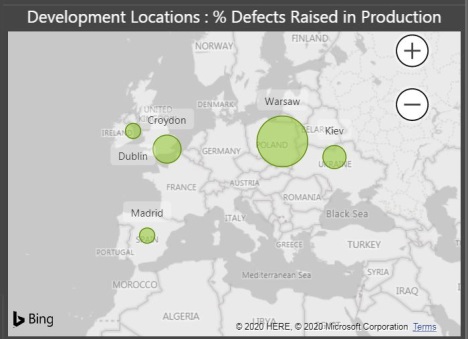
This is a relatively simple, and somewhat crude, view of the offices generating software components that have recorded defects in production on client site.

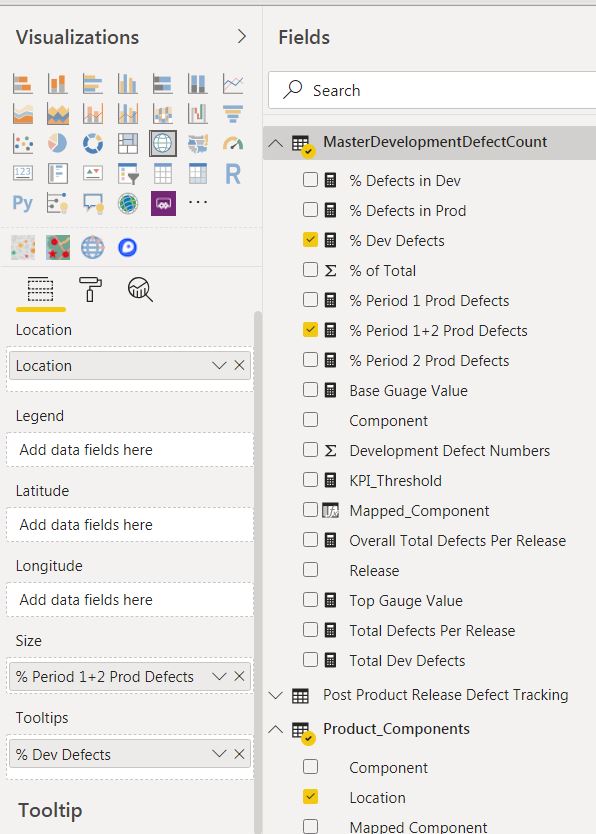
As mentioned elsewhere in this report, it is not about assigning blame but looking if there is a need to review QA processes in any given location.

Warsaw might be a case for QA process review but in reality the spread of production defects contributed by the different offices is relatively uniform.

#### Visualisation Process

The map visualisation is driven off a relationship link from the PowerBI *MasterDevelopmentDefectCount’* data source to the reference data for ‘Product\_Components’.





Defects numbers are linked to city offices through the locations for each component development.

In the company, software development for a particular component is kept within a distinct geographical location.

#### Why This Type of Visualisation?

As this is an assessment of work from different software development sites within the company it was logical to represent this data in a map visual

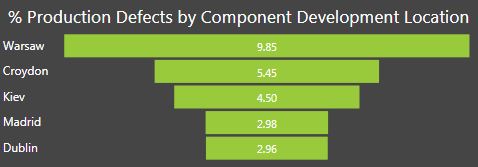
#### Relation To Other Tiles in Dashboard.

Selecting different ‘Release’ version(s), either through the Slider or by selecting a given bar in one of the charts, will update the Map graph and associated ‘bubble’ for each office.

### Tile 1.4: % of Production Defects by Development Component Locations

This tile is a compliment to the map tile (Tile 1.3) to show the relative percentage of defects found in production based on component development location.

(Each component is only developed in one location, although a location may develop multiple components).

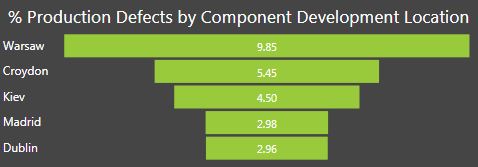


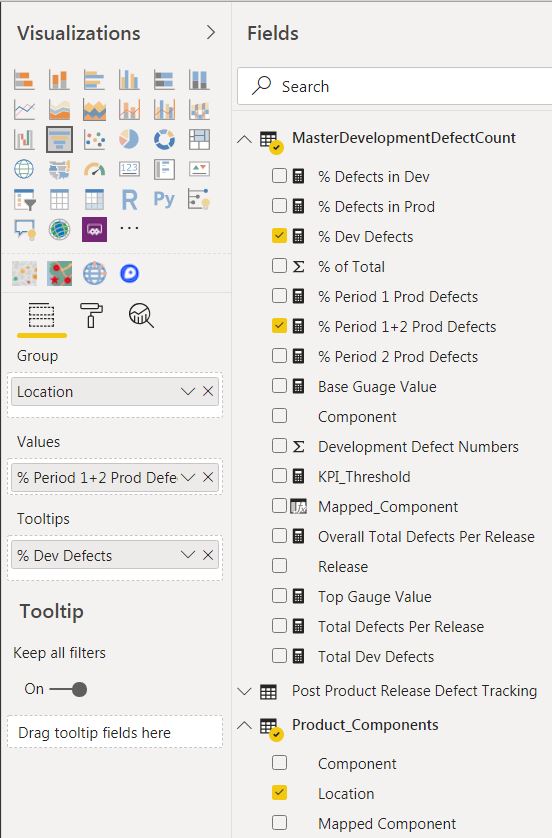
#### Purpose of Visualisation

Provide a ‘funnel’ visual to show relative percentage of defects found in production for software components developed in a given company office.

#### Visualisation Process

The tables and measures follow the same principle as those used in the map tile (Tile 1.3).





#### Why This Type of Visualisation?

It is a simple graphical aid to emphasise the date represented in the ‘bubbles’ in the tile above it in the dashboard.

#### Relation To Other Tiles in Dashboard.

Selecting different ‘Release’ version(s), either through the Slider or by selecting a given bar in one of the charts, will update the horizontal columns in the ‘funnel’.

### Tile 1.5: High Level Defect Numbers – Development and Production

This is a simple of set of numbers, within a Multi-Row Card visual, showing the overall number of defects found per Release(s), and how that compares with production numbers.



The numbers default, when the dashboard loads, to the over defect metrics for all software product releases in the project period.

#### Purpose of Visualisation

This tile provides a set of three numbers to provide a number to compliment the visuals in the bar charts in the dashboard.

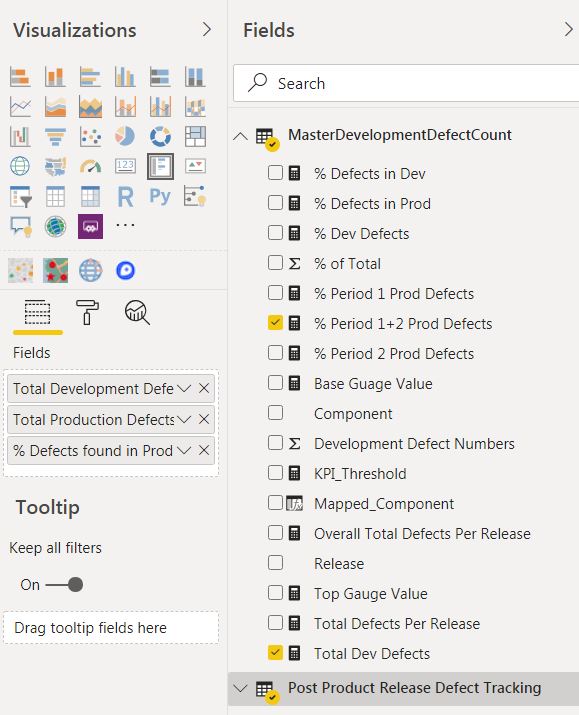
Tile 1.1 shows percentage ratios of defects, while this tile shows the actual numbers of defects driving the percentage calculations.

#### Visualisation Process

This tile directly reads the following measures from both the PowerBI *‘MasterDevelopmentDefectCount’* and ‘*Post Product Release Defect Tracking*’ data sources:

* Total Dev Defects
* Total Production Period Defects
* % Period 1 + 2 Prod Defects





The measures have been renamed on the actual tile for the purposes of presentation.

#### Why This Type of Visualisation?

This ‘visualisation’ provides some concrete numerical values to compliment the bar charts on the visualisation.

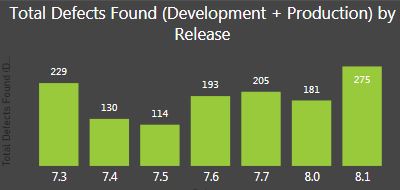
#### Relation To Other Tiles in Dashboard.

Selecting different ‘Release’ version(s), either through the Slider or by selecting a given bar in one of the charts, will update the numerical values in this Multi-Row Card visual.

### Tile 1.6: Defect Numbers Found Per Release

This tile provides a quick visual on the scale of defects found in each product release. It combines the numbers found by the company QA process during software development, along with those found after the product shipped to client site.

(The production numbers are deliberately truncated after certain periods to avoid double counting across the company client base – see Section 2.4 in this document for details).

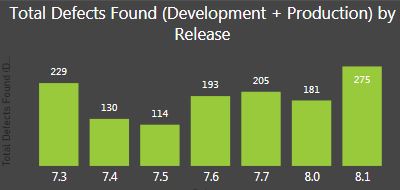


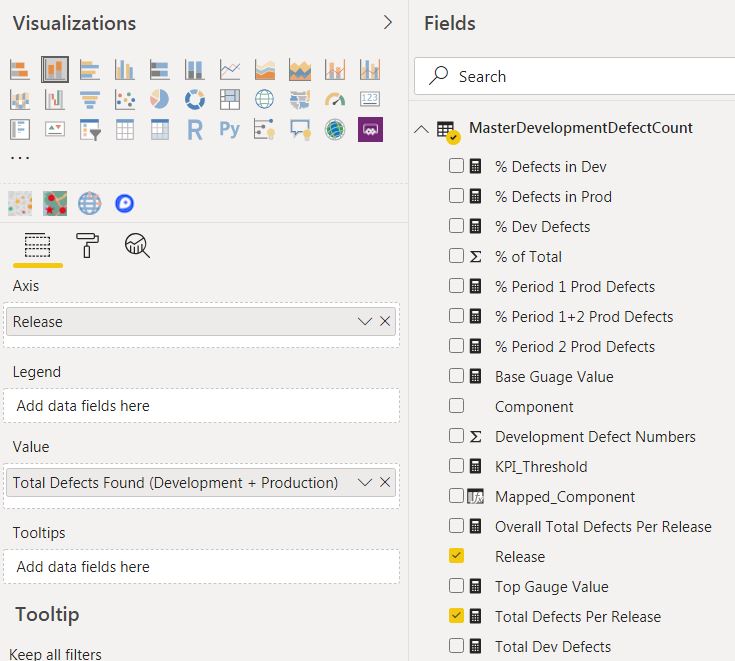
#### Purpose of Visualisation

This is a straightforward bar chart visualisation of the numbers of defects found for a given release over the development and production deployment ‘lifetime’.

#### Visualisation Process

The axis on the bar chart is determined by the ‘Release’ entry in the *‘MasterDevelopmentDefectCount’* PowerBI data source.





The size of the bars for each release is determined by the value in the measure: ‘Total Defects Per Release’.

The measure has been renamed on the visualisation to ‘Total Defects Found (Development + Production)’ for the purposes of presentation on the dashboard.

#### Why This Type of Visualisation?

This bar chart compliments the ratio comparison in Tile 1.1 and is an effective way to show the numbers of defects in a series of product releases.

#### Relation To Other Tiles in Dashboard.

Selecting different ‘Release’ version(s), either through the Slider or by selecting a given bar in Tile 1.1, will update this bar chart and highlight the given selected ‘Release’.

## Dashboard Two Design – Defect Priority Dashboard

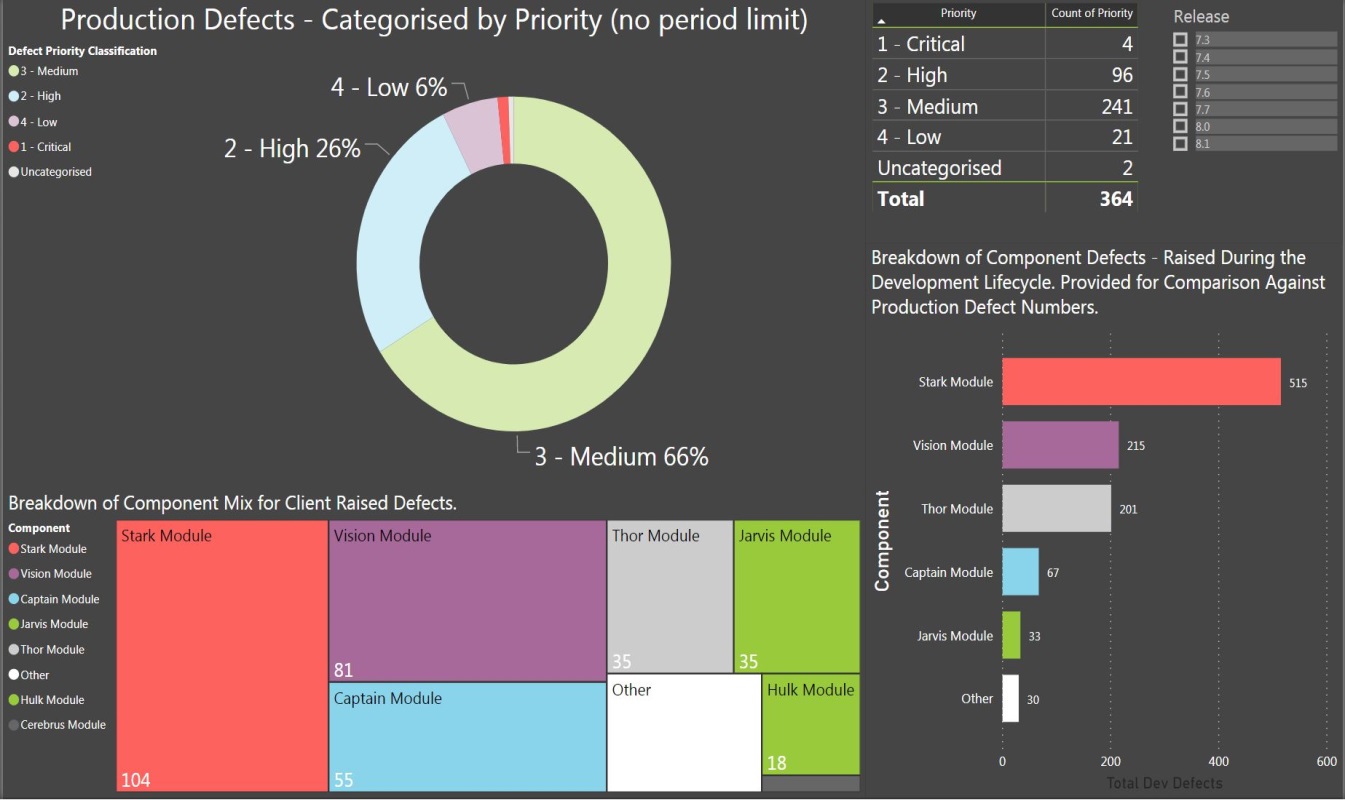
The purpose of this dashboard is to provide insight into the severity of the defects raised in production environments by clients/consultants.

**Dashboard 2: Defect Priority Dashboard**

|  |  |
| --- | --- |
| **Dashboard Component** | **Visualisation (reading clockwise from top)** |
| **Tile 1** | **Production Defects by Priority** |
| **Tile 2** | **Metrics on Defect Volumes by Priority** |
| **Tile 3** | **Defect Breakdown by Component in Development** |
| **Tile 4** | **Component Mix for Defects Found in Production** |

The second dashboard is intended to show that the significant majority of defects raised by clients/consultants in production are Priority 3 or lower.

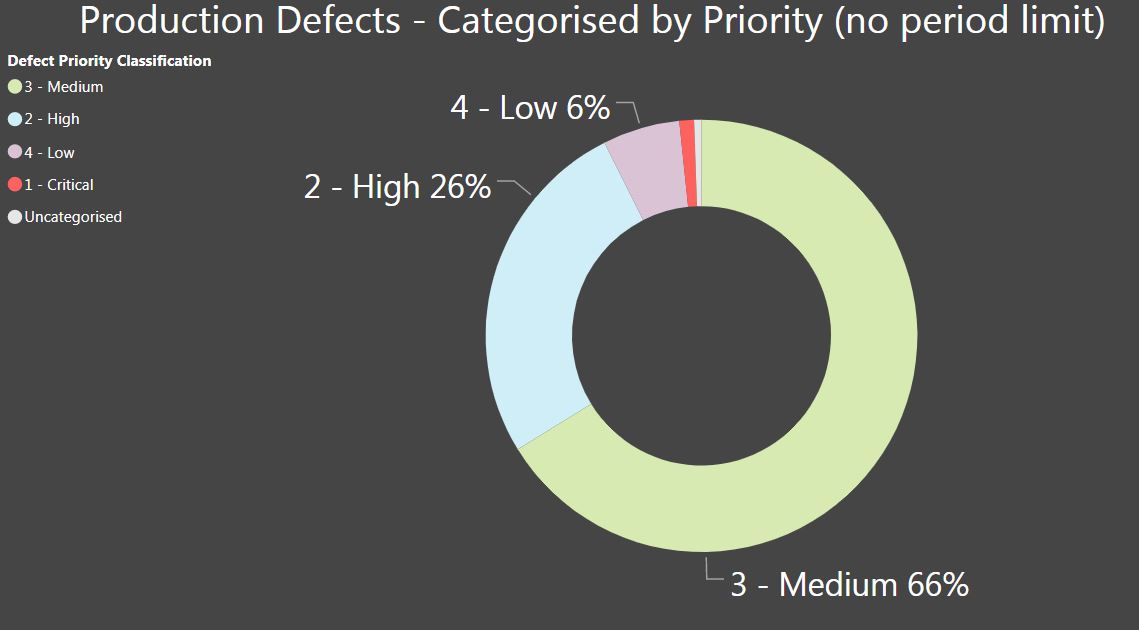
Visuals on the dashboard are also intended to show the spread of production defects across various product components and how this ratio compares to the mixture of component defects found during the development phase.



### Tile 2.1: Product Defects by Priority

Each product defect that is raised in Atlassian JIRA is assigned a priority value, which a compulsory data field.

This tile is a visual display of the breakdown of product defects raised by external consultants/clients in production, broken down by priority categorisation.



Defects marked with Priority 1 or 2 generally reflect a serious system breakdown. Defects with a Priority 3 or lower classification usually represent a problem for which there is a workaround, or may even be cosmetic.

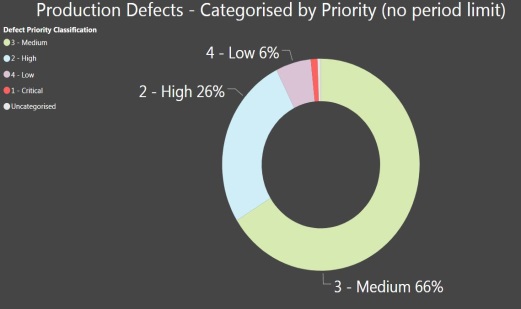
#### Purpose of Visualisation

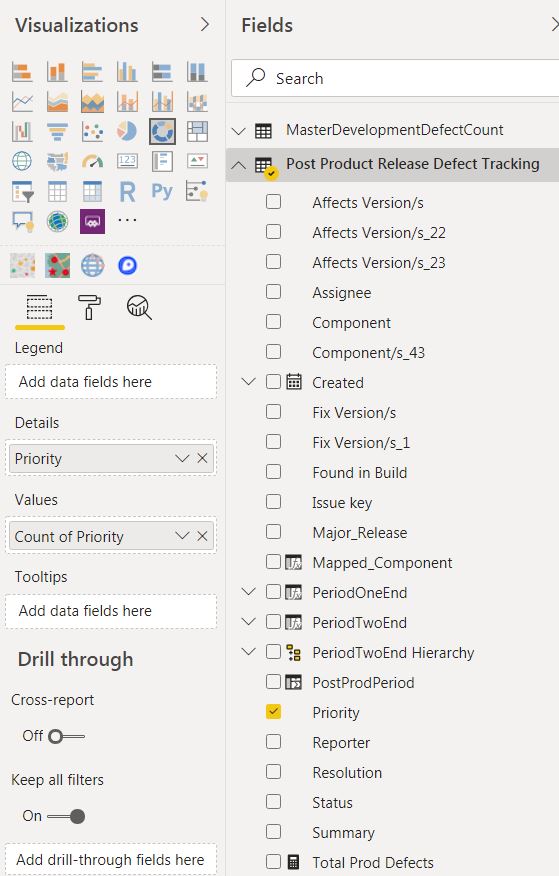
The colour segments in the PowerBI Donut Chart represent the relative proportion of defects based on their priority.

Each donut segment is labelled with it relative percentage.

#### Visualisation Process

This donut chart is based entirely from the ingested *.csv* file of production defect data, which has been read into the ‘*Post Product Release Defect Tracking*’ PowerBI data source.





Obviously, the key field being reported is the ‘Priority’ field. Different Priority categorisations are assigned a unique colour code. All rows in the data source have a Priority value; hence the data can be grouped accordingly.

#### Why This Type of Visualisation?

This tile is a logical visualisation of the five possible categorisations for a Priority value, and represents all defects recorded for the product releases tracked in this project.

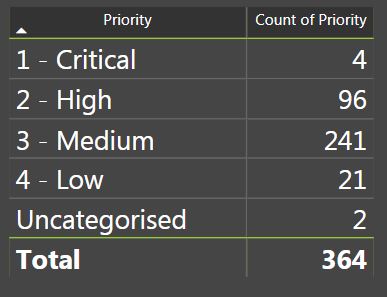
#### Relation To Other Tiles in Dashboard.

This tile will filter and update, and likewise be filtered in return, the other tiles on this dashboard that present data on production defects.

The tile that displays defects found during the product development (Tile 2.4) phase will **not** be filtered by this tile.

### Tile 2.2: Metrics on Production Defects by Priority

This tile is a simple tabular visualisation to compliment the Donut Chart and display the actual volumes of defects raised during product deployments in production environments.



Why does the total value differ from product defect numbers on the ‘***Defect Analysis Dashboard***’?

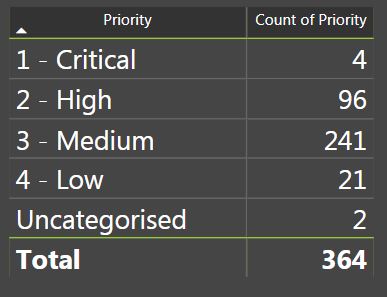
Different clients may assign different priority categorisations, even if they are reporting the same product defect. I choose to broaden the timescale of this priority analysis beyond the timeframe described in Section 2.4 because I wanted to look at an unbounded view of a defect priority analysis. Basically, the question I wanted insight on is whether the client base a whole is reporting significantly more P3 or lower issues in production.

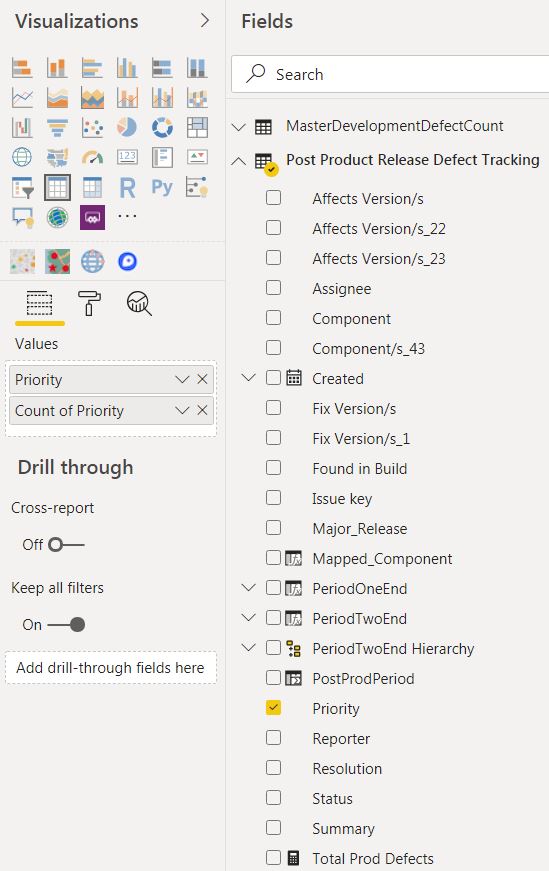
#### Purpose of Visualisation

The Donut chart in Tile 2.1 shows the proportion of production defects broken down by priority. This table provides a view of the actual numbers behind the proportions and allows for an easier side analysis when the dashboard user filters on ‘Release’ number.

#### Visualisation Process

The table is powered from the same data sources/fields as Donut Chart in Tile 2.1,





#### Why This Type of Visualisation?

It is easy to read the values ‘at-a-glance’ in a table when it is necessary to present a series of numbers broken down by a category, which is ‘Priority’ in this instance.

#### Relation To Other Tiles in Dashboard.

This tabular tile will filter and update, and likewise be filtered in return, the other tiles on this dashboard that present data on production defects.

The tile that displays defects found during the development (Tile 2.4) phase will **not** be filtered by this tile.

### Tile 2.3: Defect Breakdown by Component in Development

This tile is different from the other tiles in the dashboard because is runs of defect data from the software development phase.



#### Purpose of Visualisation

The TreeMap in Tile 2.4 provides a graphical view of the spread of production defects across various product components.

This tile (Tile 2.3) is a horizontal bar char visualisation of the breakdown of defects in the development phase.

It is provided on this dashboard to provide a colour coded and size based representation of components that generate the most defects in the development phase and the purpose is to show where defects raised in production for a given component/release seem out of phase with development defect volumes.

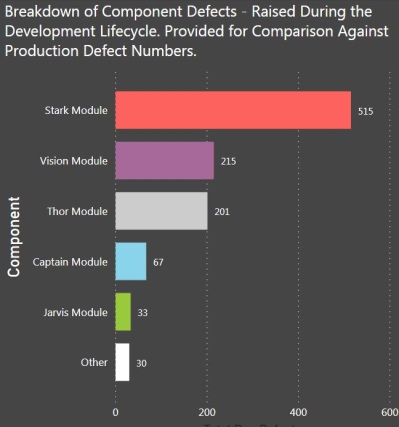
Ideally, the proportion of defects raised on a component in the development phase should strongly correlate with the volume found in production.

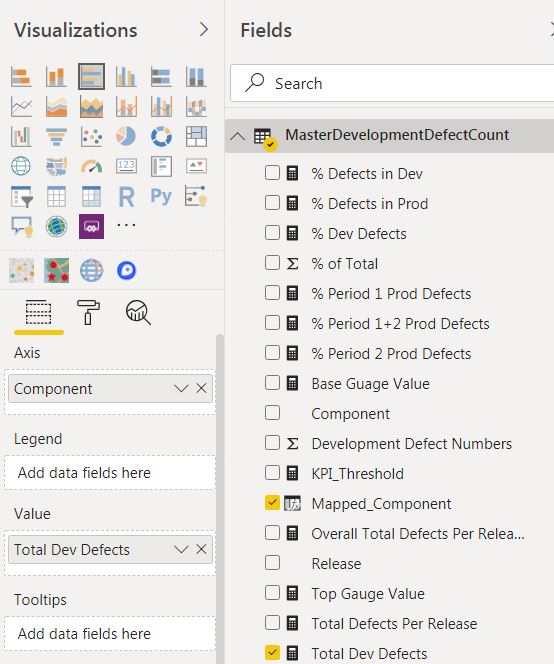
My logic is that if a component is generating a low volume of defects in the development phase but a large number in production, then some further investigation is warranted.

(The product developed by the company is made up from the integration of various sub-components developed across different geographical locations).

#### Visualisation Process

The horizontal bar chart reads the PowerBI *MasterDevelopmentDefectCount’* data source and counts the measure ‘Total Dev Defects’ and groups the added column ‘Mappred\_Component’.





‘Mapped\_Component’ is an added column because I wanted to restrict and simplify the component names being returned within the JIRA *.csv* file. Some of the component names used in Production are inaccurate and I have applied a LOOKUPVALUE routine to tidy up the data for this visualisation.

The description of the column has also been changed within the tile for presentation.

#### Why This Type of Visualisation?

The size of the horizontal bars make it relatively easy to compare against the TreeMap in Tile 2.4 in terms of component defect magnitudes.

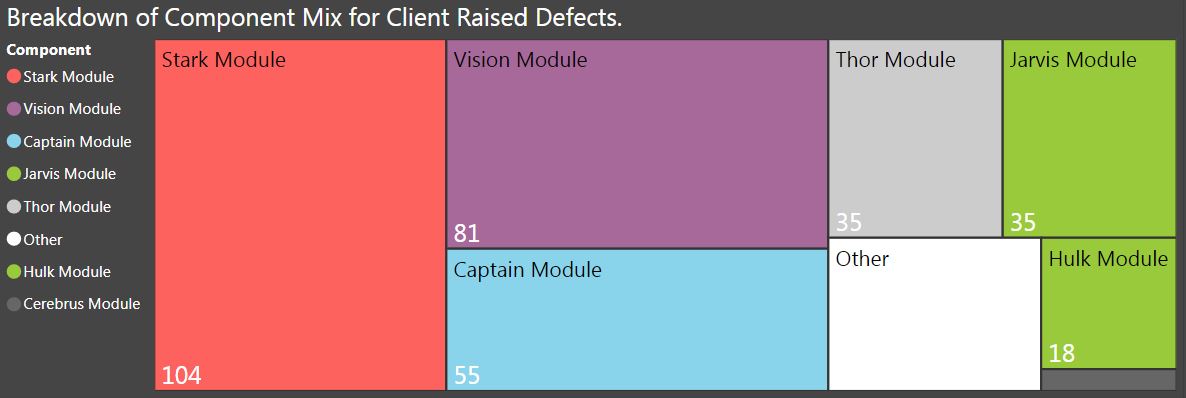
The colour coding in Tile 2.3 and Tile 2.4 have been aligned to aid visual inspection of component defect data.

#### Relation To Other Tiles in Dashboard.

This tile is independently of the production defect data in the other tiles in this dashboard. It is not filtered by changes in the other tiles.

### Tile 2.4: Component Mix for Defects Found in Production

This tile is a visualisation of the relative spread of defects found in production based on component type.



A high volume of defects found for a particular component in production does not, in itself, indicate a problem.

For instance, the components named ‘Stark Module, ‘Captain Module’, and ‘Vision Module’ are much larger those named ‘Cerberus’ and ‘Hulk’. It would be expected that these modules generate a larger volume of defects, commensurate with the scale of development work that they involve.

However, if the proportions of defects found in Production shift significantly between releases, or are noticeable out of step with the volumes of defects found in development then further investigation may be required.

#### Purpose of Visualisation

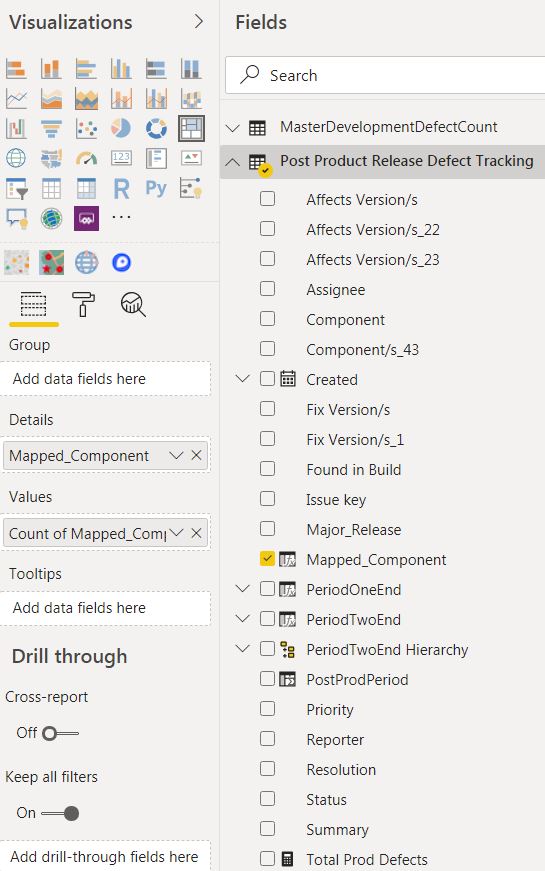
A product consists of multiple components. The TreeMap provides a view of the relative proportion of defects generated by each major component.

Those modules generating more defects will gain more prominence in the TreeMap, particularly as the user selects between different individual releases.

#### Visualisation Process

The TreeMap visualisation is generated by reporting on the added ‘Mapped\_Component’ field in the ‘*Post Product Release Defect Tracking*’ PowerBI data source.





#### Why This Type of Visualisation?

The relative contribution to overall defects numbers in production by each component can be easily represented in a TreeMap.

The rectangles in the TreeMap adjust as the user selects between releases and the defect proportions change for each component.

Selecting Priority categorisations in the Donut Chart in Tile 2.1, will help the user see that the proportional spread of defects is fairly even across components (the filter highlights the relevant number in each ‘box’ in the TreeMap).

#### Relation To Other Tiles in Dashboard.

This TreeMap tile will filter and update, and likewise be filtered in return, the other tiles on this dashboard that present data on production defects.

The tile that displays defects found during the development (Tile 2.4) phase will **not** be filtered by this tile.

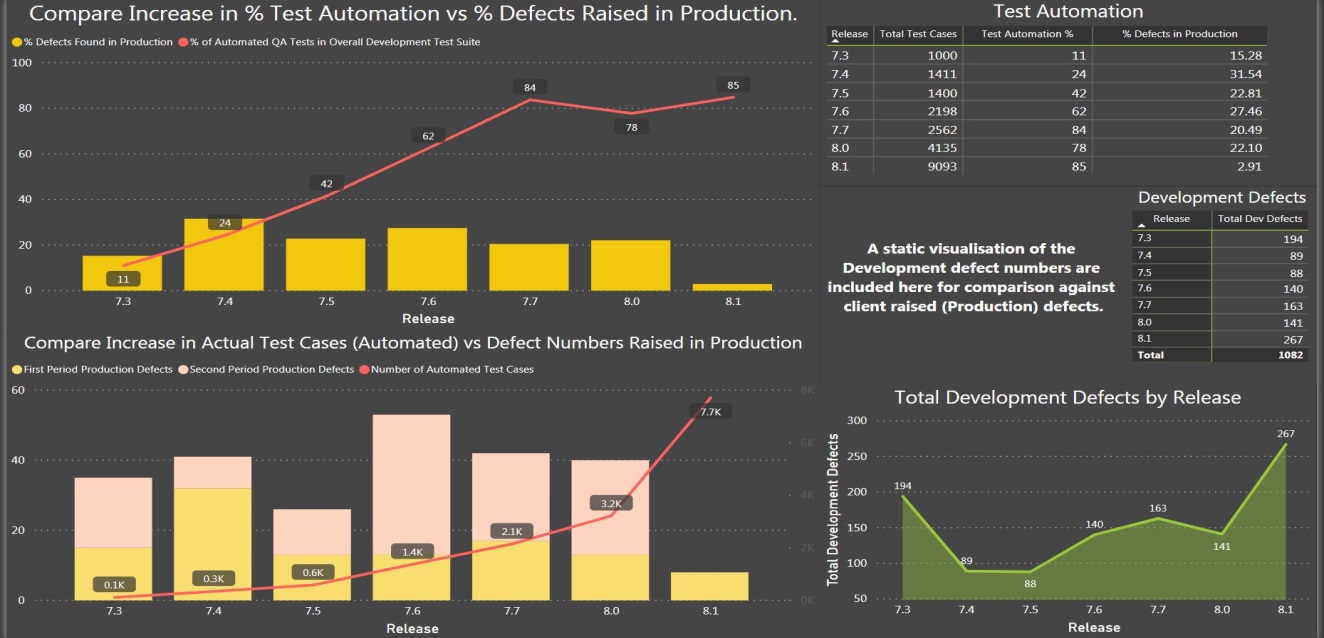
## Dashboard Three Design – Test Automation Dashboard

The purpose of this dashboard is to show the impact that increased investment in QA test automation technologies is having on defect trends, both during development and on the numbers of defects being found in production.

**Dashboard 3: Test Automation Dashboard**

|  |  |
| --- | --- |
| **Dashboard Component** | **Visualisation (reading clockwise from top)** |
| **Tile 1** | **% Increase in Test Automation vs % Production Defects** |
| **Tile 2** | **High Level Test Case Automation Metrics** |
| **Tile 3** | **High Level Development Defect Metrics (Table)** |
| **Tile 4** | **High Level Development Defect Metrics (Area)** |
| **Tile 5** | **Increase in Test Automation Numbers vs Production Defect Numbers.** |

The third dashboard shows an overlap of the increase in test automation over time, for each recent product release, and the change in defects reported. There are slight, but not spectacular improvements to be seen in the visuals.

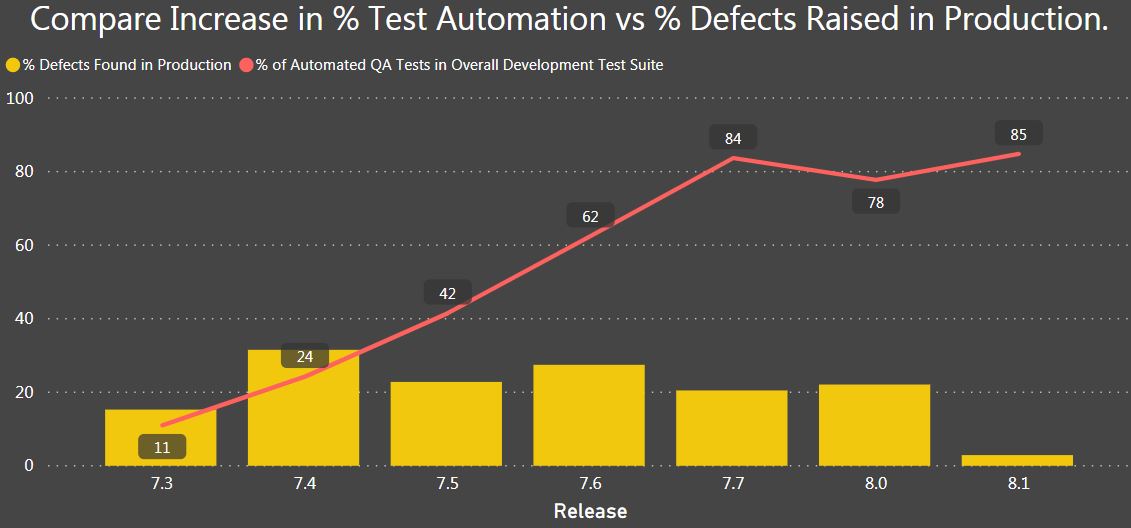


The metrics for the latest product release are based on relatively little data, as this release has not had much time to penetrate the marketplace.

### Tile 3.1: Increase in % Test Automation vs % Production Defects

This tile is a Line and Clustered Column chart that shows the increase in the levels of test automation within the companies QA process over the releases in the project period.

This information is overlaid as a line on top of a bar chart showing the percentage of defects found in production during the ‘lifetime’ of a product.



#### Purpose of Visualisation

This tile is intended to determine if the increase in the volume of software testing through test automation with each release is having an impact on product quality (measured by the ratio of defects found in production).

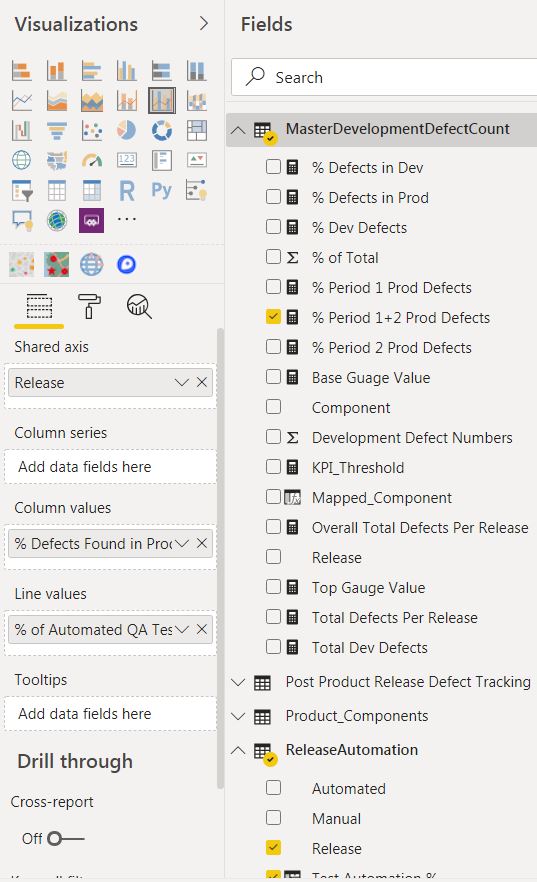
As each release is developed the volume of testing increases because more automated Selenium tests can be executed in each development cycle.

Excluding the incomplete data on the last release, these improvements in product quality are noticeable, if not overly remarkable.

#### Visualisation Process

The chart axis is the ‘Release' value and the line in the chart is read directly from the added column ‘Test Automation %’, which is a measure added to the ‘Release Automation’ data source in PowerBI.





The bar charts are read from the measure created to calculate the percentage ratio of production defects per release.

#### Why This Type of Visualisation?

The line and the bar columns are both measures in percentage and the overlay allows for a direct visual comparison.

Arguably we would like to see a more striking inverse relationship between the line and the columns – to demonstrate that as the company invests more in test automation technology there is a significant drop in production defects and hence product quality.

That said, the visuals in the chart provide some reason for optimism with future releases.

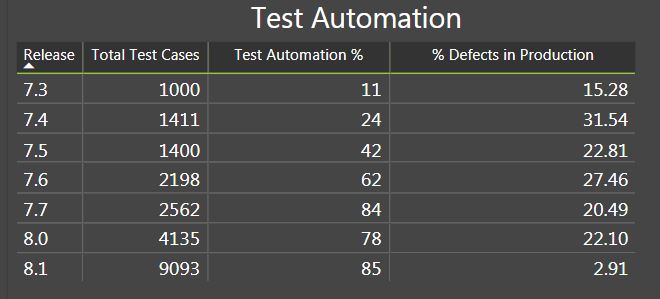
#### Relation To Other Tiles in Dashboard.

Selecting a column in the bar chart of this visualisation impacts on the other production defect bar chart tile (3.5) and the Test Automation table (3.2), but not the development defect visuals.

### Tile 3.2: High Level Test Automation Metrics

This tile is a tabular representation of the actual test cases executed for each release during the development cycle, along with the percentage of those tests that were automated.

Once a test is automated it is permanently added to the release test suite. Hence with each new release there is a growing rollover of test automation from previous releases.



The percentage of production defects are added to elaborate on the bar columns in Tile 3.1

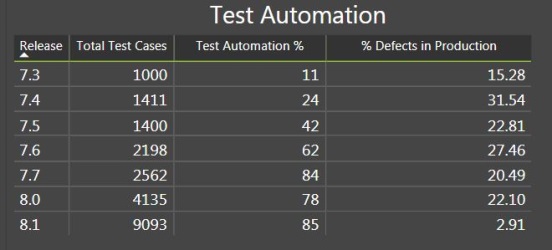
#### Purpose of Visualisation

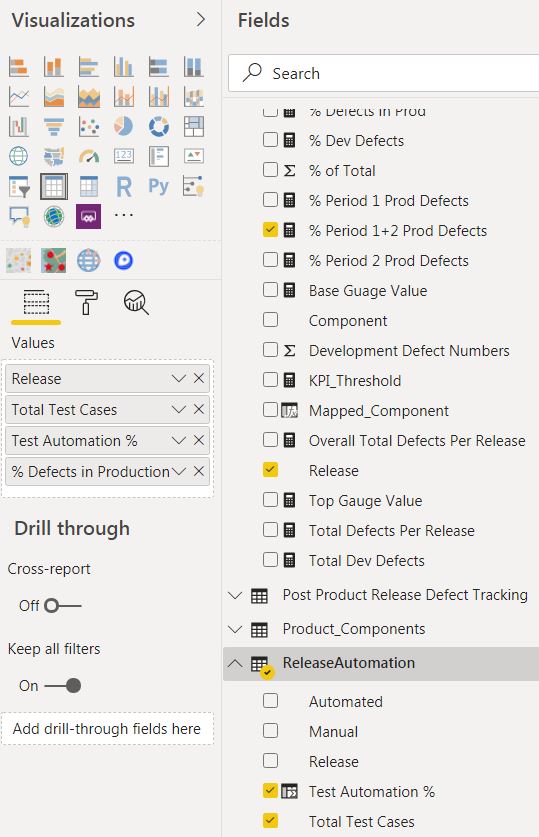
This tile provides a quick way for the consumer of the dashboard to see the numbers involved in test automation for each release.

The tile compliments the visual date in Tile 3.1.

#### Visualisation Process

The table is generated from test case numbers (per release) and the added column ‘Test Automation %’ in the ‘ReleaseAutomation’ data source.





#### Why This Type of Visualisation?

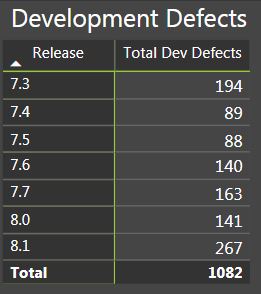
A table visual is the most effective way to display ‘volume of per release’ numeric data on test case numbers.

#### Relation To Other Tiles in Dashboard.

Selecting a line entry in the table of this visualisation impacts on the other production defect bar chart tile (3.5) and the Test Automation table (3.2), but not the development defect visuals.

### Tile 3.3: Development Defect Numbers (Table)

The data in this tile has been produced in other forms on the other dashboards, but it is included here again to show if product quality, as measured by defects found in the development phase is being effected.



#### Purpose of Visualisation

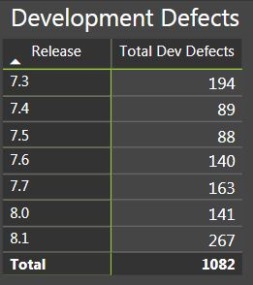
Does an increase in test automation improve quality, as measured by the number of defects found in the development phase?

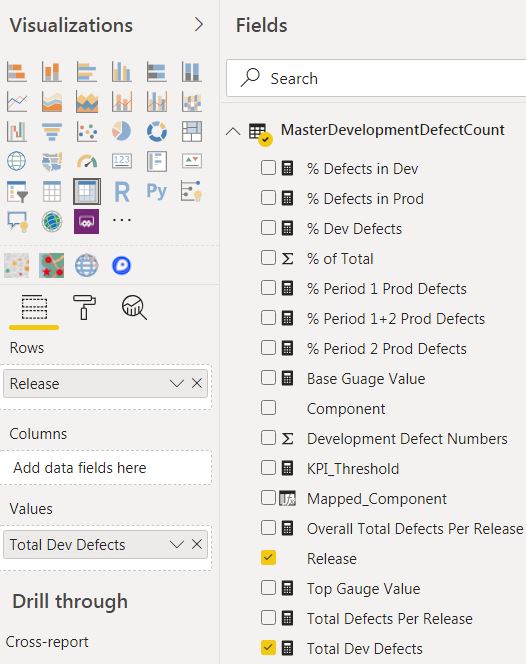
This can be seen as an inverse proposition to the production defect analysis. The QA process in development should possibly be finding more defects before the product ships, at a point when they are cheaper for the company to fix.

The tabular data does seem to suggest that the QA process is generally improving with each release in terms of catching defects earlier.

#### Visualisation Process

The table is relatively simple in construction, as it reads the ‘Release’ value and the added measure ‘Total Dev Defects’ from the *‘MasterDevelopmentDefectCount’* PowerBI data source.





#### Why This Type of Visualisation?

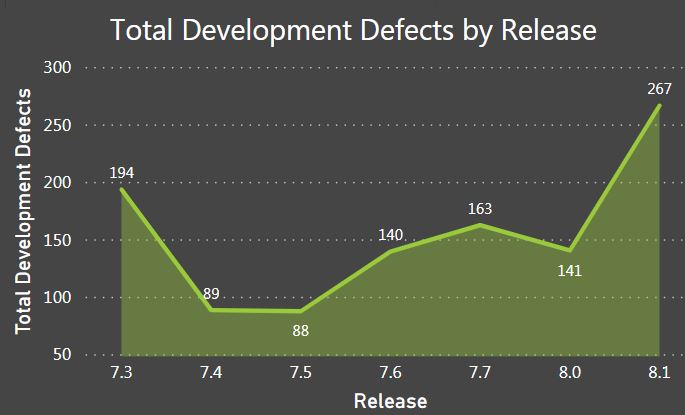
The tabular form is an easy way to represent the sub total of defect numbers for each release.

#### Relation to Other Tiles in Dashboard.

It is a tabular representation of the area chart in Tile 3.4. It does not active any dynamic filtering on other visuals in this dashboard.

### Tile 3.4: Development Defect Numbers (Table)

This tile is an area graph that visually compliments the tabular data in Tile 3.3.



It is a repetition of the data in Tile 3.3 but it provides a visual representation of the generally upward trend in defect detection by the companies QA process during product development.

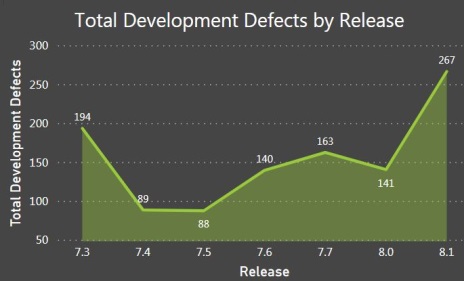
#### Purpose of Visualisation

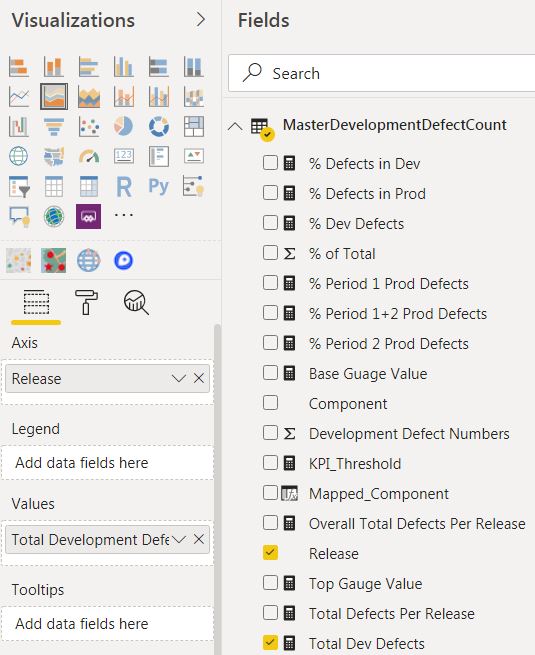
To show, in a more visual manner, that increased test automation is improving test defect detection.

This area chart does not explicitly reference test automation data but it is placed firmly in the context of a dashboard that reports on the evolution of company QA processes.

#### Visualisation Process

The area chart reads the ‘Release’ value and the added measure ‘Total Dev Defects’ from the *‘MasterDevelopmentDefectCount’* PowerBI data source.





#### Why This Type of Visualisation?

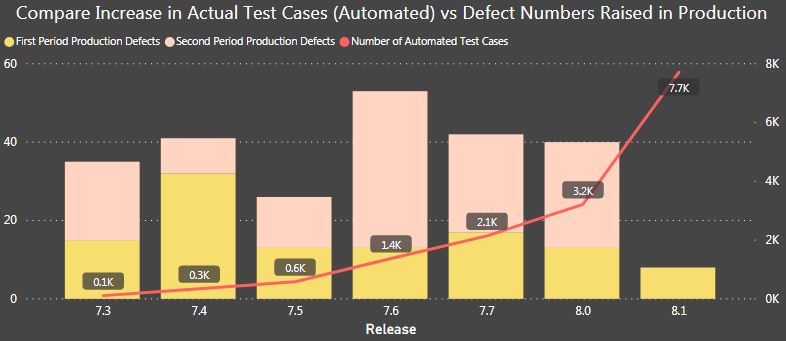
The area chart is a visual compliment to the tabular data in Tile 3.4.

#### Relation To Other Tiles in Dashboard.

The tile is a visual representation of the table chart in Tile 3.3. It does not active any dynamic filtering on other visuals in this dashboard.

### Tile 3.5: Compare Increase in Test Case Numbers to Production Defect Numbers

This tile provides a view that is similar to the line and bar chart in Tile 3.1 but this time it overlaps the volume of actual test cases, as the number increases – through automation – with each release.



The amber and pink stacks in the columns for each release refer to the two periods during which production defects are tracked against each release.

The left Y-Axis tracks the production defect numbers for the bar columns.

The right Y-Axis tracks the numbers of test cases executed for each release (in thousands).

#### Purpose of Visualisation

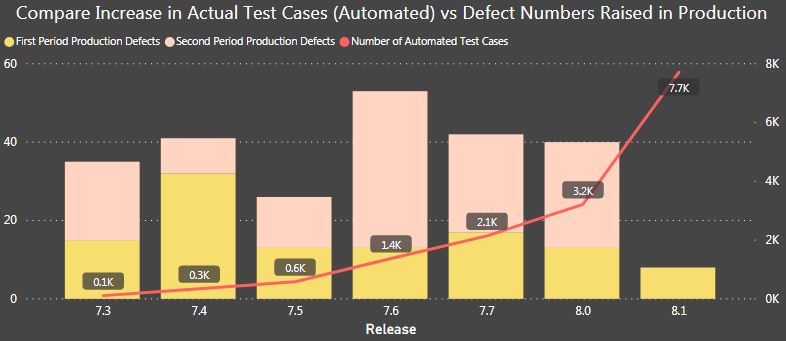
This tile compliments the data in Tile 3.1 by showing again that the later product releases are showing a slight improvement in product quality as the scale of test automation increases.

#### Visualisation Process

The chart reads from the following measures created on the ‘*Post Product Release Defect Tracking*’ data source, which is read from the JIRA .csv extract of production defect data;

* TotalPeriodOneProdDefects
* TotalPeriodTwoProdDefects

These measures build up the yellow and pink segments in the bar columns to show a more granular view of the production defect numbers.



The line is read from the ‘Automated’ and ‘Release’ fields in the ‘ReleaseAutomation’ data source in PowerBI.

#### Why This Type of Visualisation?

This tile provides a similar overlap as in Tile 3.1. It is a useful way to counterpoint the increase in the volume of test automation against the number of defects found in production, per release.

#### Relation To Other Tiles in Dashboard.

Selecting a column in the bar chart of this visualisation impacts on the other production defect bar chart tile (3.1) and the Test Automation table (3.2), but not the development defect visuals.

# Conclusions

## Interpretations from the Quality Assurance Visualisations

These dashboards have been created around a question – are the gradual changes in the Quality Assurance process within the company having an impact on software product quality?

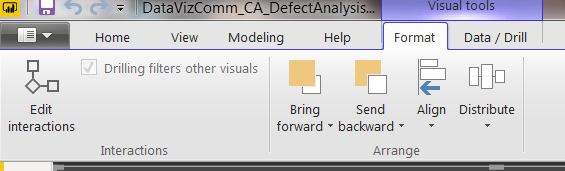
In building these dashboards, I have personally made the following observations;

1. This is ‘real world’ data, and I will be evolving these dashboards further for use in regular presentations on the company QA process. However, domain context is essential, and it is part of the reason for the length of this report. Dashboards on a topic like Covid-19 would be immediately comprehensible to the general public. These dashboards require introducing a significant element of ‘scene setting’. I believe this is acceptable because the presentation of these dashboards will always be to an audience that understands the business context.
2. The data set is not large. It generally reflects actual working metrics but even the production data is less than 2000 rows. Is there enough information to infer meaningful insights? I believe that as the data reflects activity trends over 3+ years for seven separate software product releases we can see useful patterns emerge.
3. The information from the visualisations is interesting but the actual message is not starkly obvious. Are the changes in company Quality Assurance process making the product ‘better’? The answer from the visualisation would appear to be ‘probably’. For me, one key objective for the dashboards is to have a framework for ongoing visualisation of future product releases. The company is likely to keep releasing this type of product for the foreseeable future. Hence, there is a reasonable expectation that these dashboards will help show if more defects continue to be captured during development, and fewer defects are materialising in client production environments.
4. Although there is a range of visualisations across all the dashboards, the same development defect data is often repeated in different forms. I used this approach because I believed it was useful to re-iterate that the production defect data (volume and priority) and the test automation data should be seen in context to behaviour in the development phase.

## Challenges in the Quality Assurance Data Visualisations

My list of key challenges with the generation of these dashboards would be;

1. As described in Section 6.1, the volume of development data is not large. I believe it still represents enough information volume to be meaningful.
2. The source of the development defect data was fragmented across spreadsheets, created at different times over the last three years. It was necessary to pull the information from various fileservers and re-format before the work described in this project could even begin. Validating the data in advance was also a tedious process. However, a major benefit that this project will deliver is removing the dependency on disparate Excel spreadsheets and replacing it with a central PowerBI platform.
3. In the PowerBI Query Editor there is a difference in approach to loading some of the separate release data spreadsheets as compared to the technique shown in DBS lectures. I was attempting to build a structure that could be converted into a PowerBI template with embedded parameter settings to read in new file locations and release numbers. This made the data ingestion phase of the project more complex but my intension is to revisit this objective to build a PowerBI template from this CA work in the near future.
4. Adding the new columns to data sources could have been avoided if I built a data model in a relational database and used this as the information source with additional SQL generated ‘Views’. This may be an approach I will revisit in the future with an Azure based RDBMS on the same company platform as our hosted PowerBI application.
5. Creating the new columns in PowerBI involved the use of a number of in-built Data Analysis Expressions (DAX). The syntax is slightly different in places to functions I would be familiar with for similar operations in Excel. This required an element of additional research, but that was a rewarding activity in itself.
6. Not every visualisation on a dashboard needs to be filtered by changes in the other visualisations. Early drafts of the ‘Priority’ dashboard were set up so that changes in how the user viewed the ticket priority information activated filters in the development defect graphs. These changes made no obvious sense (there is no priority information tracked within the development data). It was necessary to use the options within PowerBI to ‘Edit Interactions’ and remove those dynamic filters.



# References

## The Visualisation Theme

The colour scheme, and minor layout changes to the axis, title, and labels, were imported from the Microsoft custom theme marketplace.

The online marketplace can be found here : <https://community.powerbi.com/t5/Themes-Gallery/bd-p/ThemesGallery>.

The particular theme used is named ‘Nowalls Analytics’, which can be downloaded from this [location](https://community.powerbi.com/t5/Themes-Gallery/Nowalls-Analytics/td-p/772564).