[cover sheet 1 of 2]

2

ASSIGNMENT 2

**DEFECT ANALYSIS REPORT**

CSE 6329 -- SOFTWARE MEASUREMENT AND QUALITY ENGINEERING

Professor Dennis J. Frailey

**Fall, 2019**

NAME: **Goutami Padmanabhan**

ID Number: **1001669338**

Grader Comments:

|  |  |  |
| --- | --- | --- |
| **Grading Template (student should not write on this page)** | | |
| **Section 1 – Introduction and Overview** | | |
| \_\_\_\_\_ (/9) | 1.1 \_\_\_\_\_ Purpose of This Report (1)  1.2 \_\_\_\_\_ Structure of Analysis Tool (4)  1.3 \_\_\_\_ Data Collection Overview (2)  1.4 \_\_\_\_\_ Summary of Analyses and Graphs (2) | (9 points) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Section 2 – Measures, Graphs and Analysis** | | | |
| **Section 2.1 – Post Release Quality** | | | |
|  | i (3 points)  **Overview**  (Purpose / Question / Definition / Collection) | ii (3 points)  **Graph** | iii **Analysis** & Discussion (6 points) |
| 2.1.a \_\_\_\_\_/12 |  |  |  |
| 2.1.b \_\_\_\_\_/12 |  |  |  |
| 2.1.c \_\_\_\_\_/12 |  |  |  |
| 2.1.d \_\_\_\_\_/12 |  |  |  |
| **Section 2.2 – Current Quality** | | | |
|  | i (3 points)  **Overview**  (Purpose / Question / Definition / Collection) | ii (3 points)  **Graph** | iii **Analysis** & Discussion (6 points) |
| 2.2.a \_\_\_\_\_/12 |  |  |  |
| 2.2.b \_\_\_\_\_/12 |  |  |  |
| 2.2.c \_\_\_\_\_/12 |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Section 3 – Summary and Recommendations** | | |
| 3.1 \_\_\_\_\_\_/3 | Summary |  |
| 3.2 \_\_\_\_\_\_/4 | Recommendations |  |

|  |  |
| --- | --- |
| \_\_\_\_\_\_\_\_\_/100 | 🡸 **Total Assignment Grade** |

**Defect Analysis Report**

1. **Introduction**
   1. **Purpose of This Report**

This report shows the results of analyzing three years of defect data on our released products. The purpose is to gain a greater understanding of the quality levels of our released products and to determine whether there is any correlation between software quality and other factors such as the programming language used, the development process used, or the time when the product was developed. This report also shows the measurements of various aspects of the Software project quality thereby giving a good insight on the software metrics. It helps us to understand the post-release defect data and provides detailed information to make critical decisions on quality improvement.

* 1. **Structure of Analysis Tool**

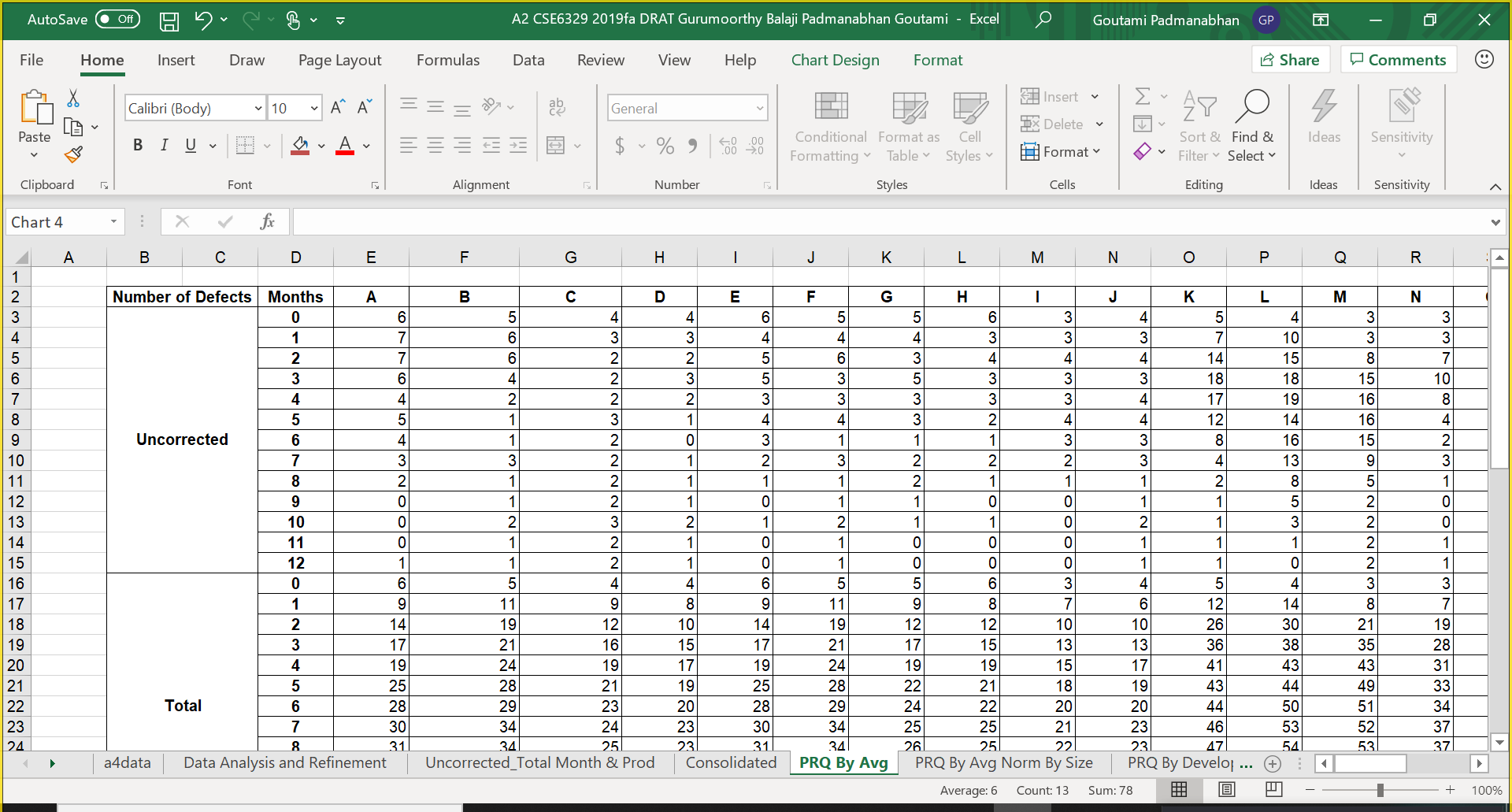
In order to analyze defect data, we have created a workbook that contains refined data required for creating tables and graphs. We have created twelve different worksheets with their corresponding graphs from the data analyzed and derived.

Each graph compares two parameters such as No. of defects and Time period (Month wise).

**1.2.1. Post-Release Quality Average for all products, Normalized by Size:**

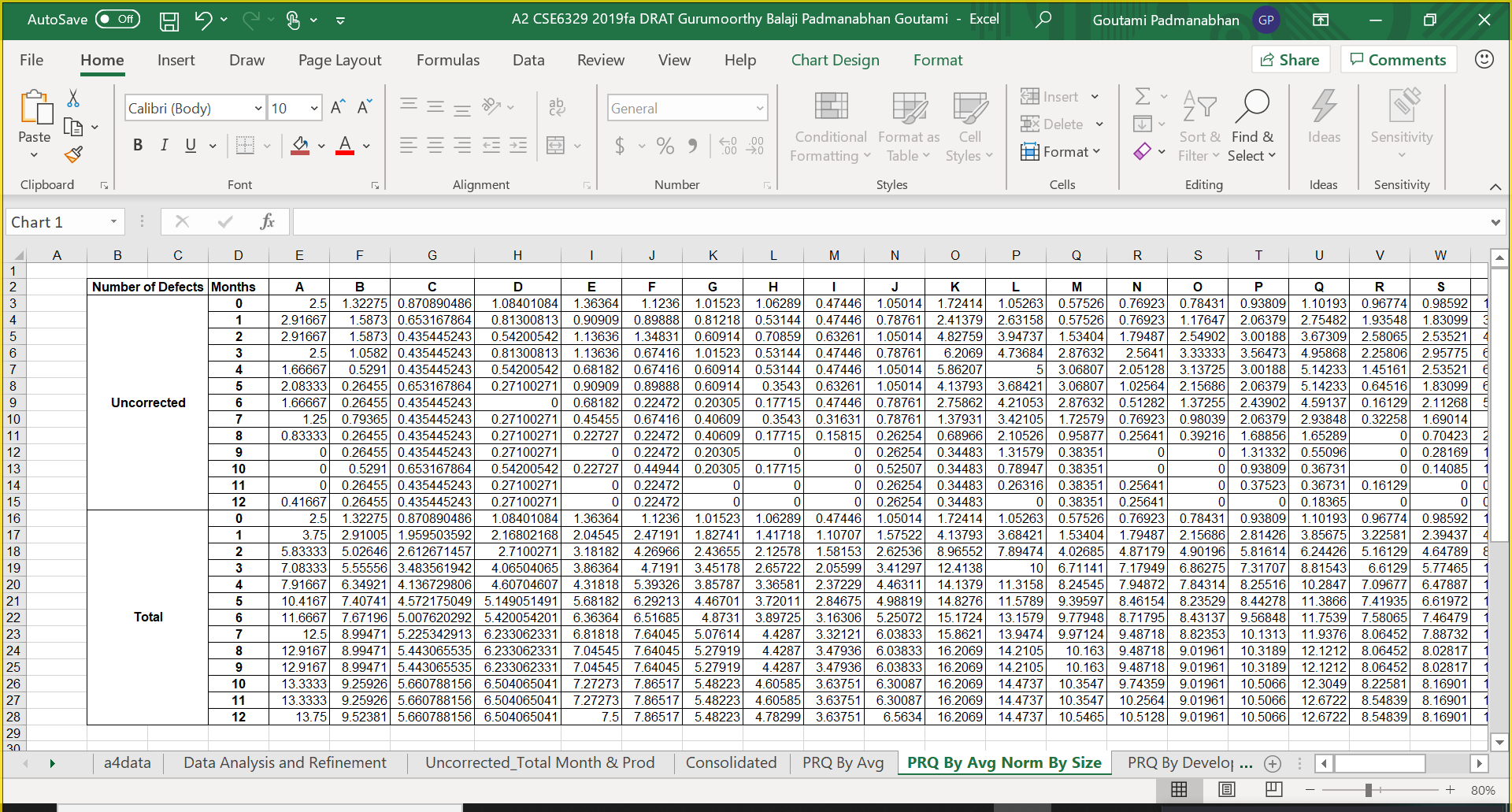
**PRQ By Avg Worksheet:**

This worksheet shows the number of uncorrected and total new defects for each product every month post release. The average of both uncorrected and total new defects is computed for each month for all the products released in that month.



**PRQ By Avg Norm by Size Worksheet:**

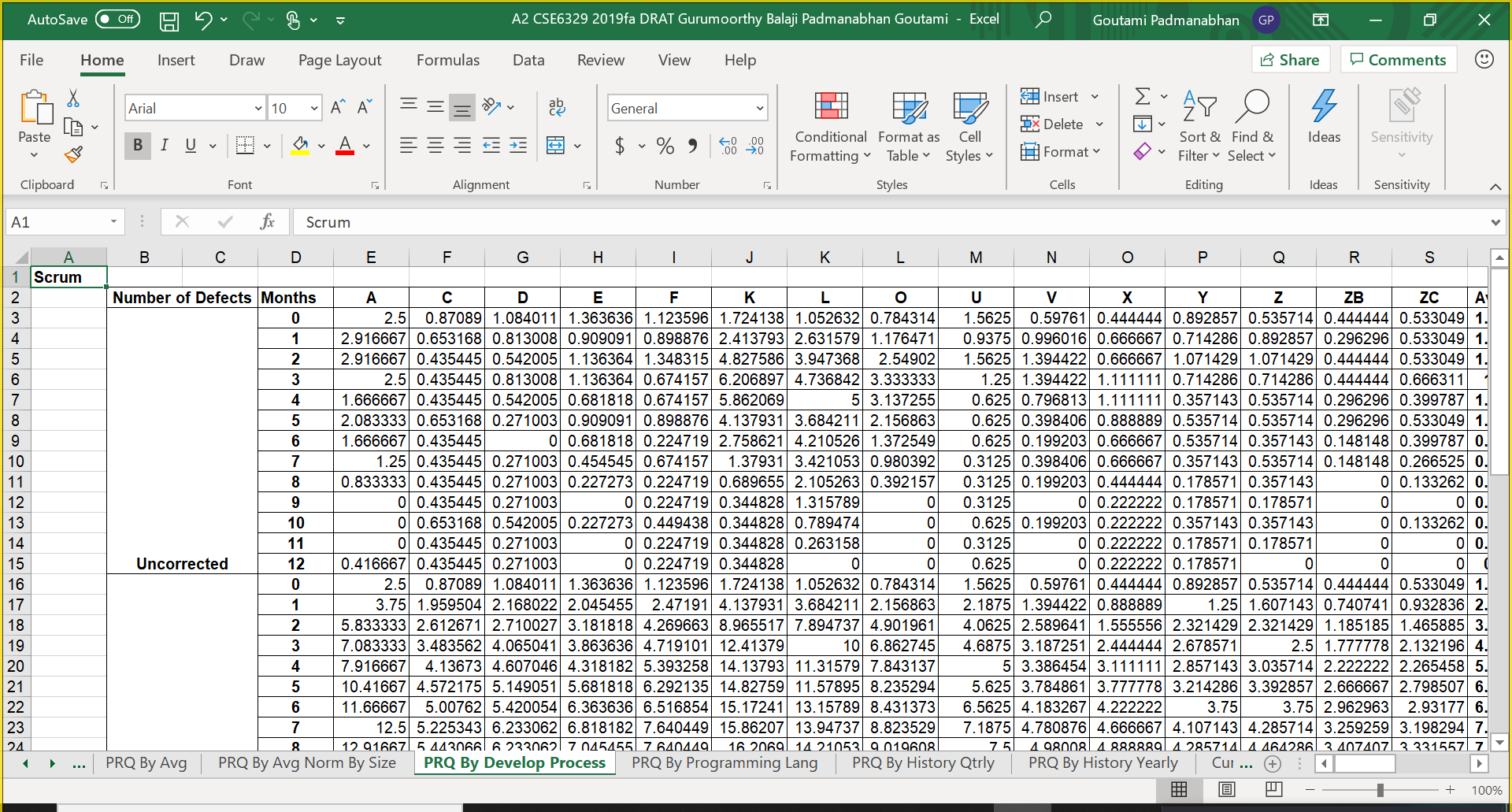
The below worksheet contains the number of uncorrected and total new defects per 1000 lines of code for each product every month post release which is normalized by the respective product size. The average normalized by size is calculated as shown.

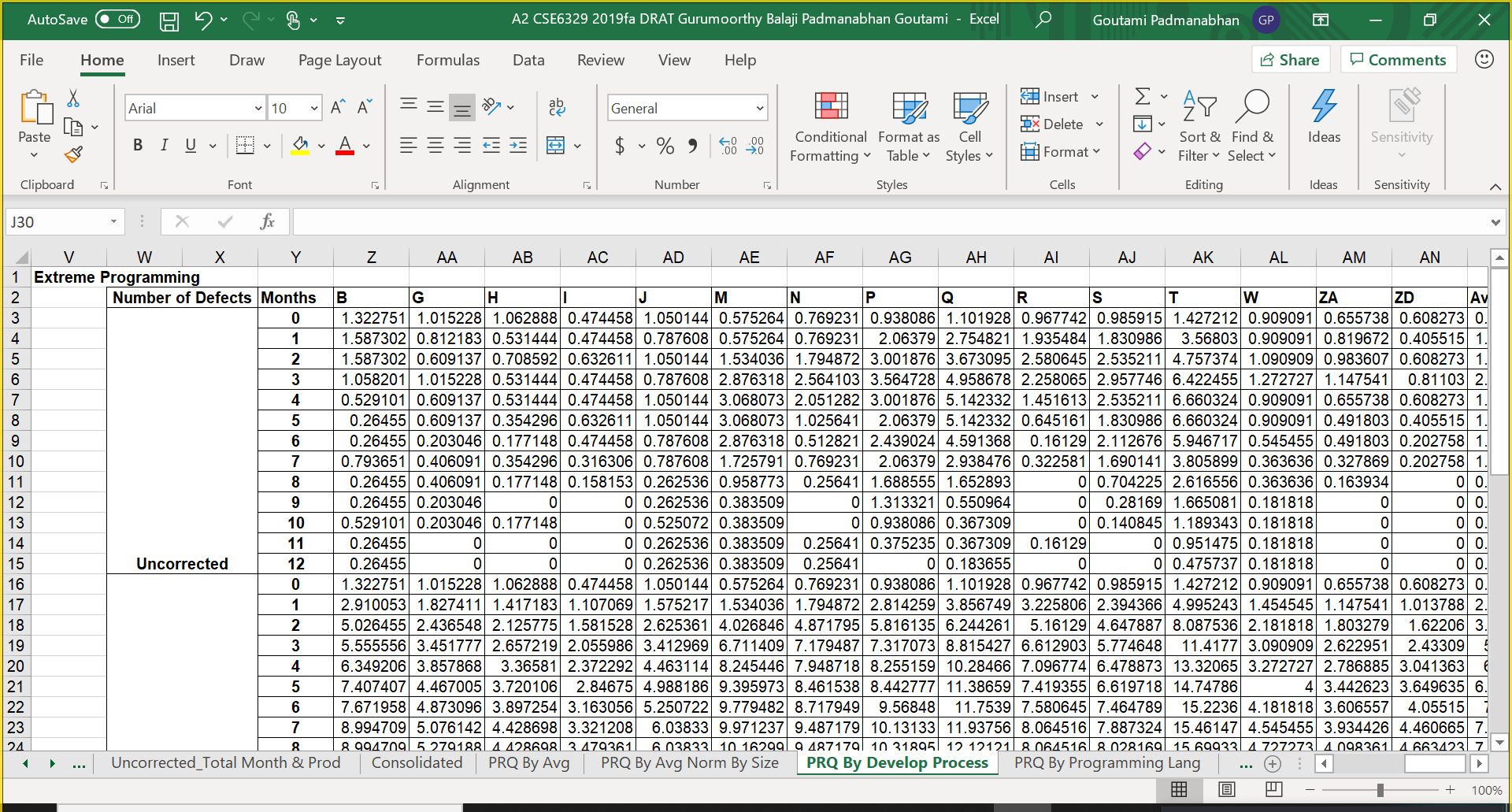


**1.2.2. Post-Release Quality Average for all products, By Development process:**

**PRQ By Develop process Worksheet:**

The below two worksheets contain the number of uncorrected and total new defects per 1000 lines of code for each product every month post release based on the development processes viz. Scrum and Extreme Programming. It is normalized by their respective product size. The averages normalized by size are calculated as shown.

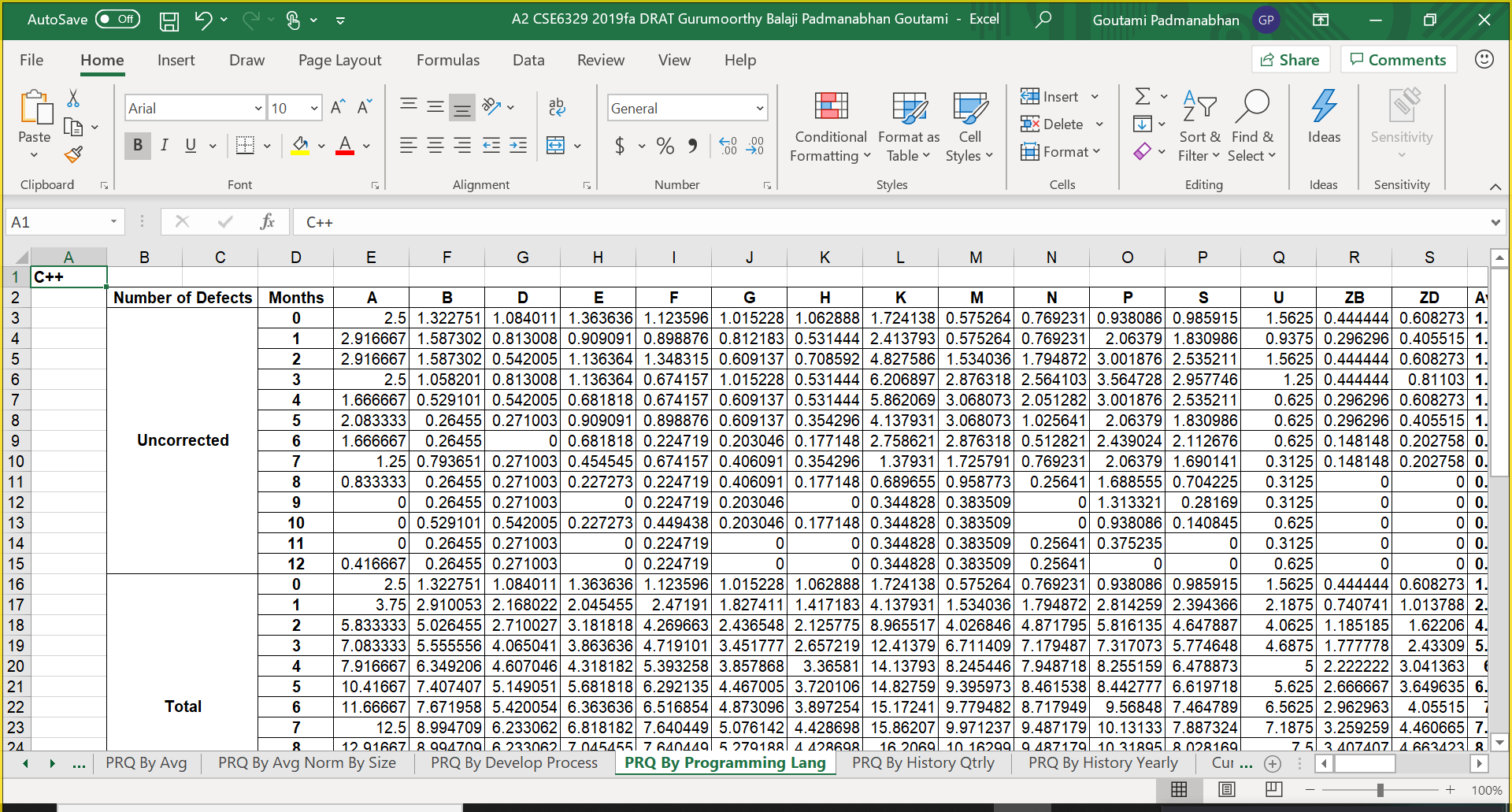


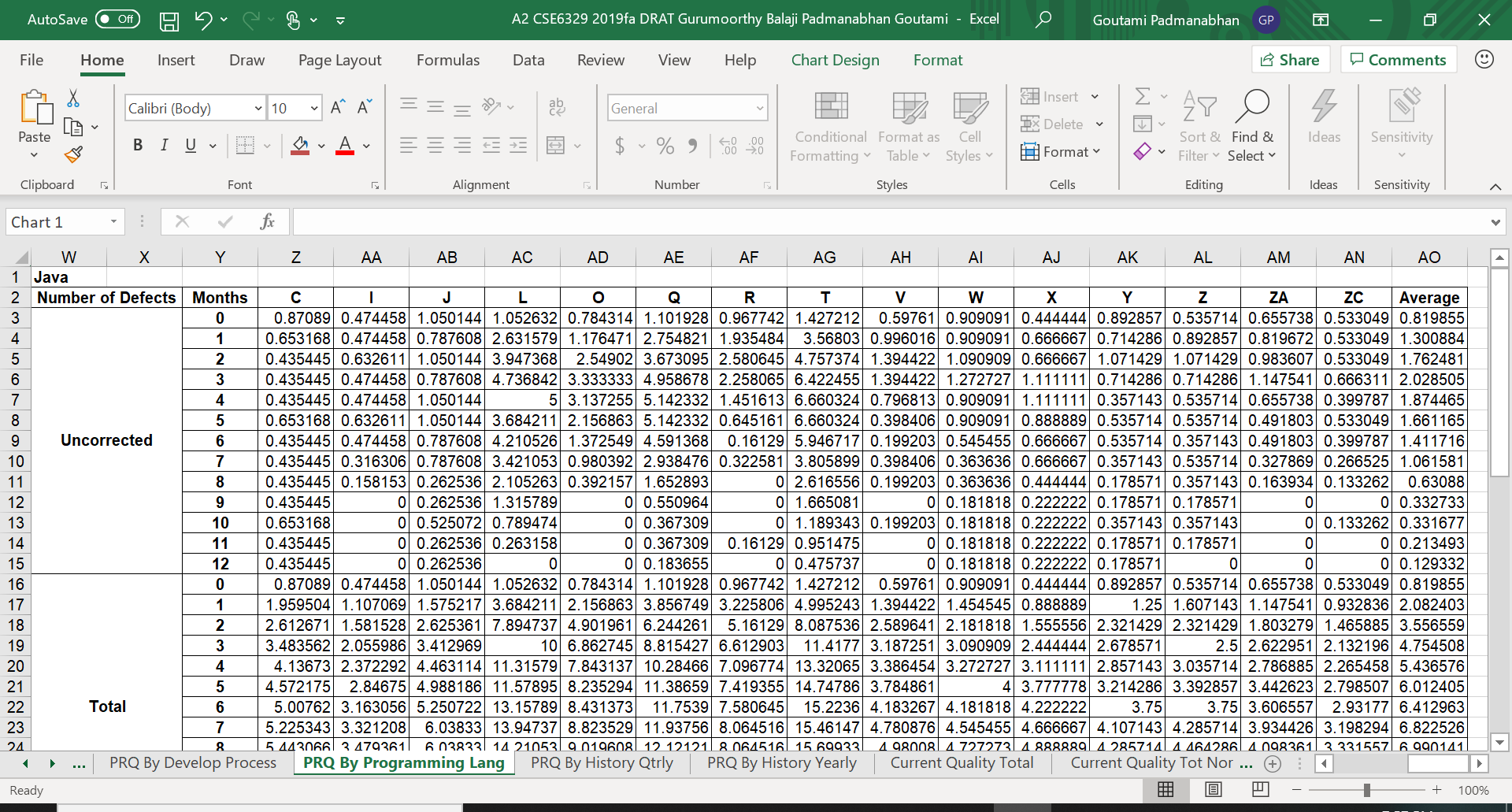


**1.2.3. Post-Release Quality Average for all products, By Programming Language:**

**PRQ By Programming Language Worksheet:**

The below two worksheets contain the number of uncorrected and total new defects per 1000 lines of code for each product every month post release based on the programming languages viz. C++ and Java. It is normalized by their respective product size. The average normalized by size is calculated as shown.

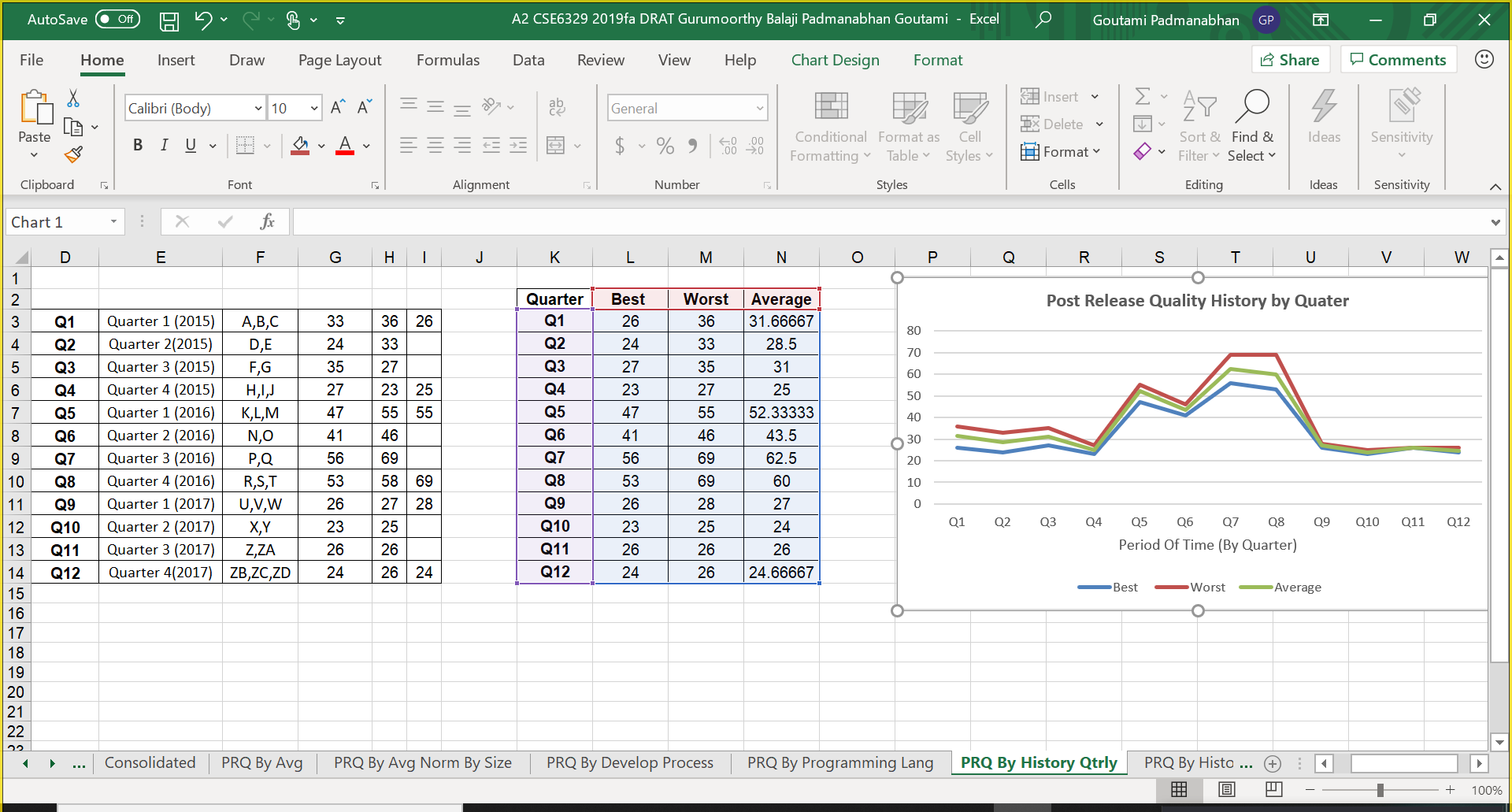




**1.2.4. Post-Release Quality History:**

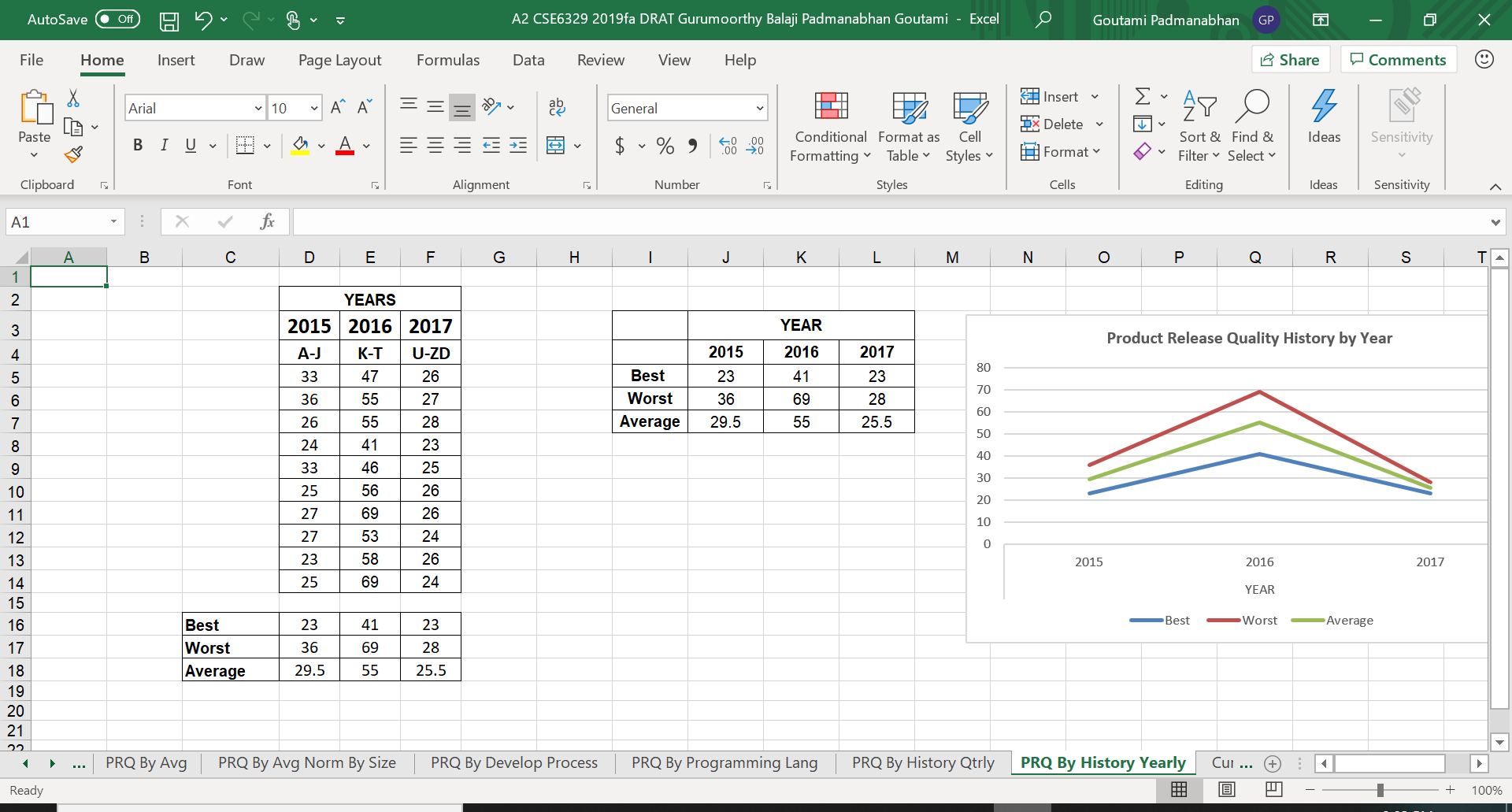
**PRQ By History Qtrly Worksheet:**

The below worksheet contains the total new defects for each product released during the three-year period for every quarter. Another table shows the Best and the Worst values of total new defects. The corresponding averages are calculated as shown.



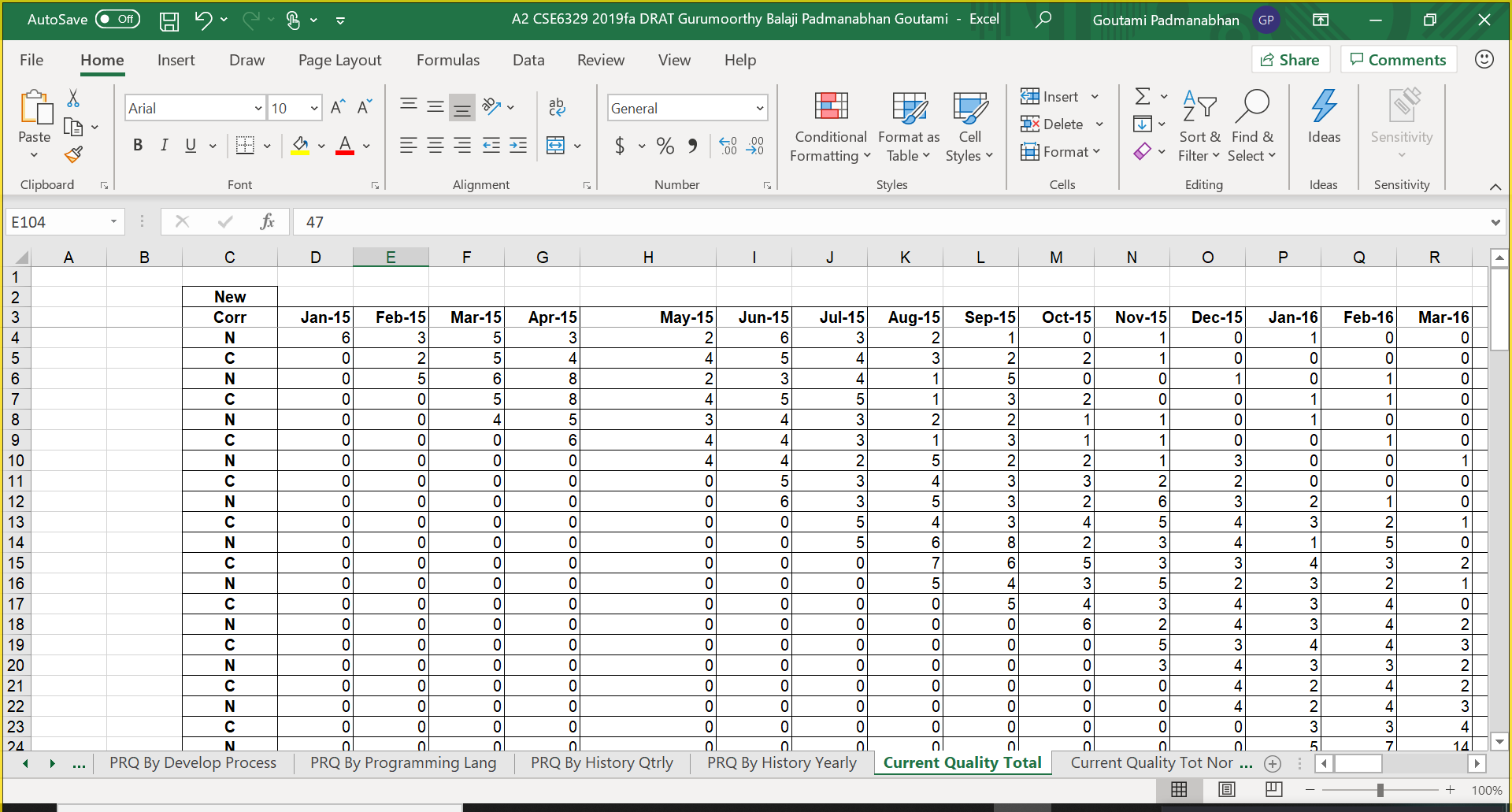
**PRQ By History Yearly Worksheet:**

The below worksheet contains the total new defects for different products released during the three-year period for every year. Another table shows the Best and the Worst values of total new defects along with their corresponding averages as shown.



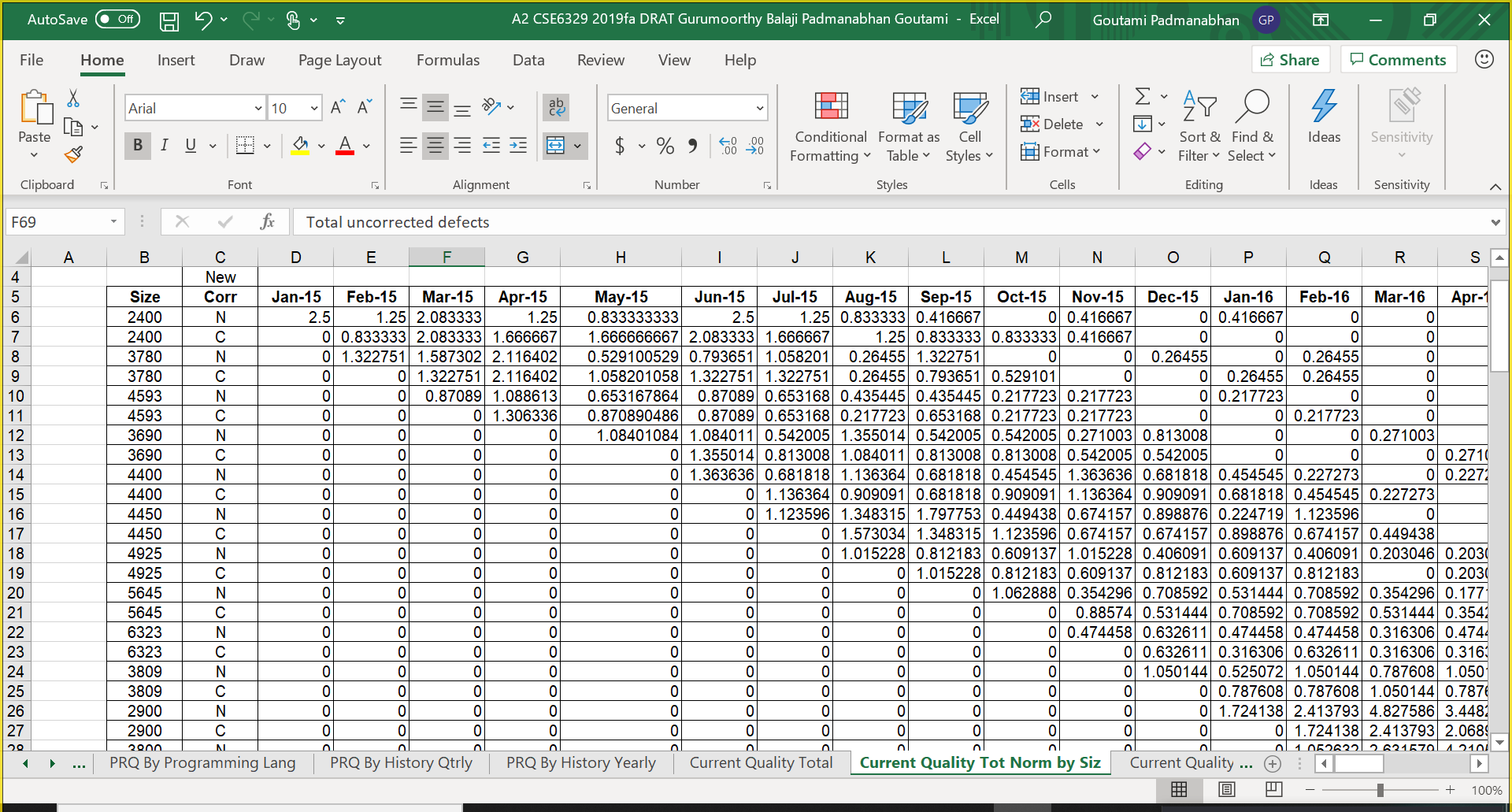
**1.2.5. Current Quality Total:**

The below worksheet contains the total uncorrected defects for all products released during the year and are still active within their first 12 months. Total uncorrected defects are calculated month wise as shown.



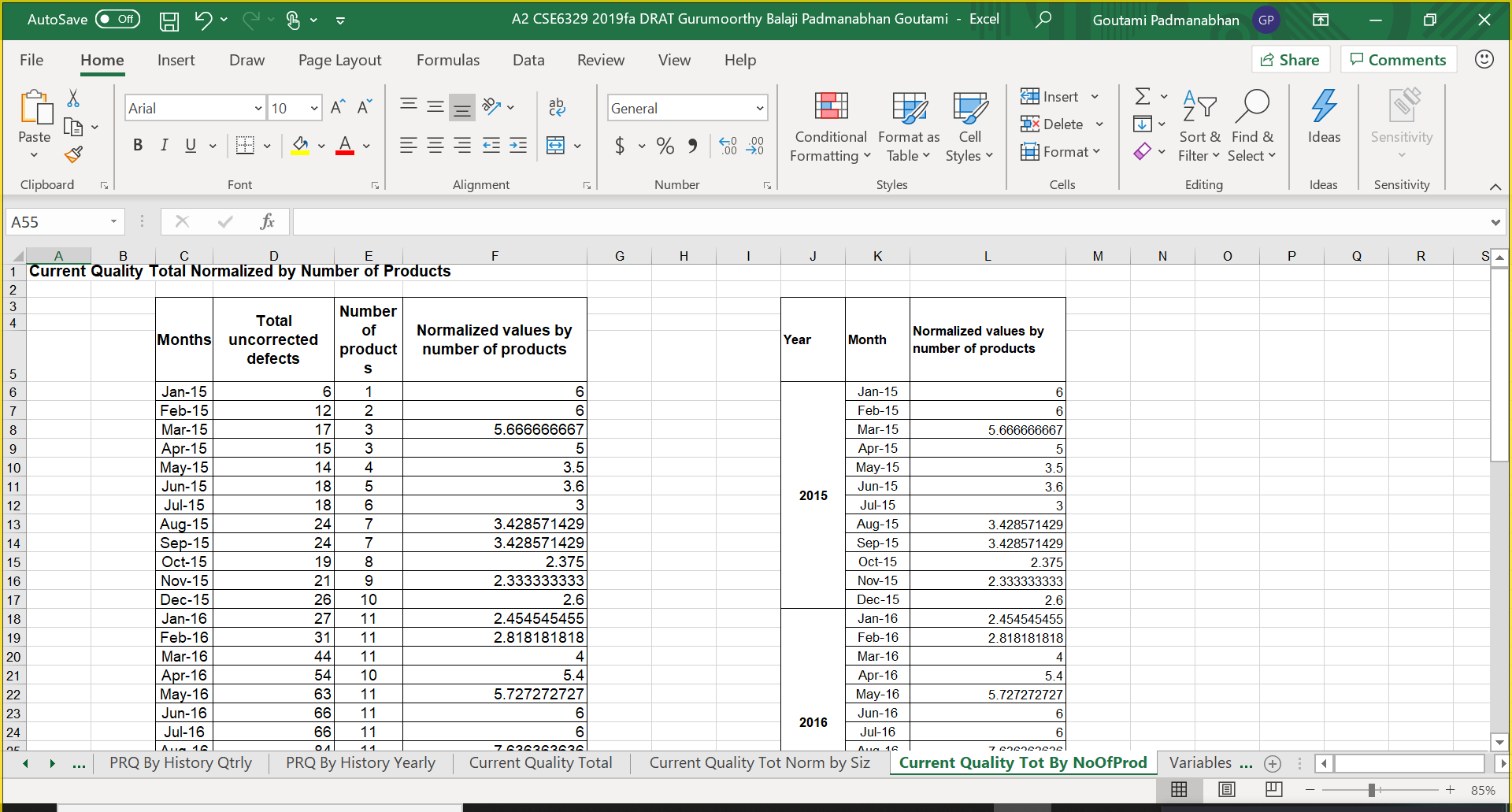
**1.2.6. Current Quality Total, Normalized by Size:**

The below worksheet contains the total uncorrected defects for all products released during the year normalized by size and are still active within their first 12 months. For each month, defects of each product are normalized by size. Total uncorrected defects are calculated month wise as shown.



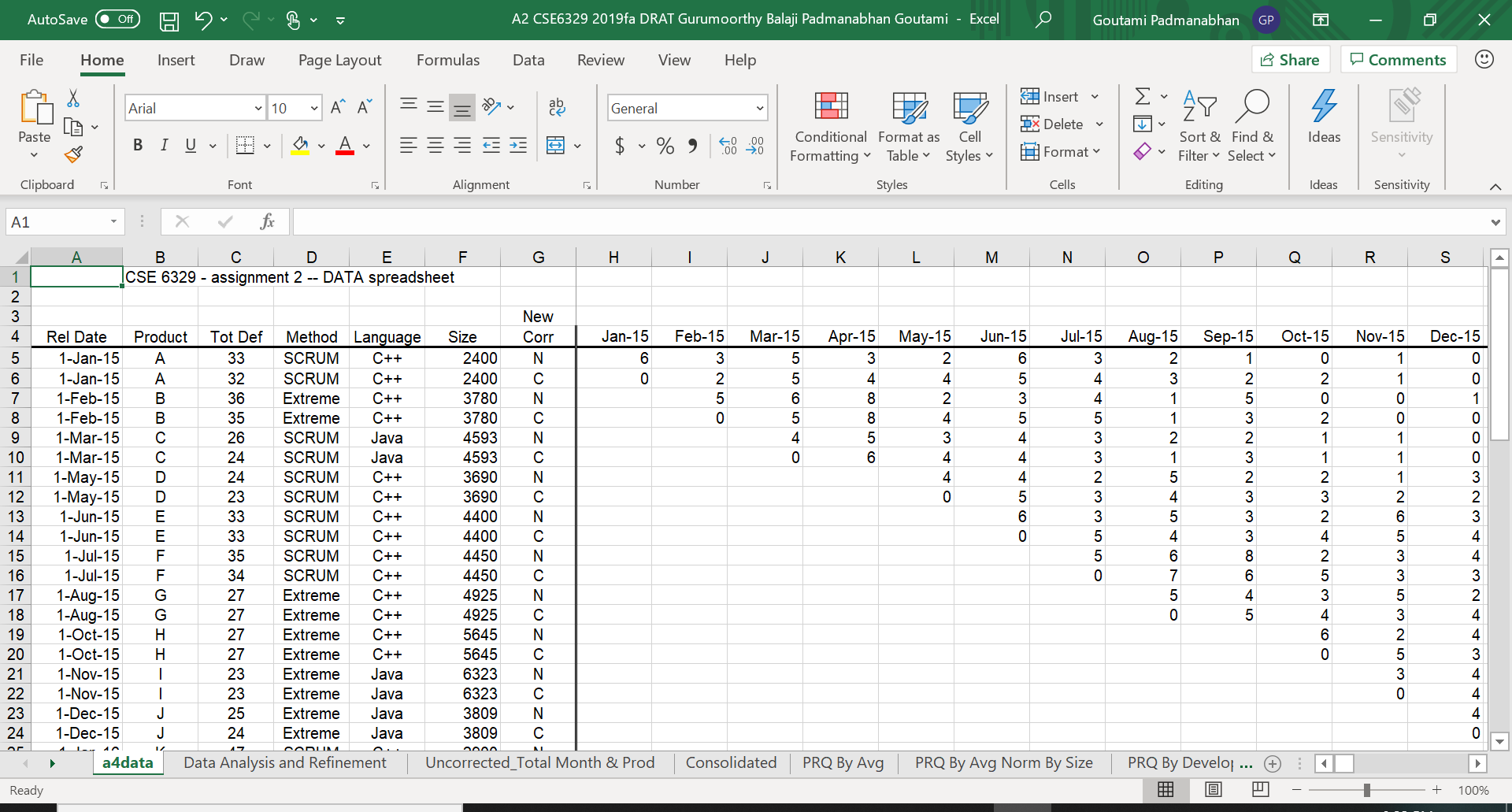
**1.2.7. Current Quality Total, By Number of Products:**

The below worksheet contains the total uncorrected defects for all products released during the year normalized by number of products and are still active within their first 12 months. For each month, defects of each product are normalized by the total number of currently active products. Total uncorrected defects are calculated as shown.



* 1. **Data Collection Overview**

The defect collected consists of the number of new defects detected and the number of defects that were corrected every month for each product. This defect data was collected over a period of 12 months post the release of each software product. The data was stored in an Excel data worksheet named ‘a4data.xlsx’ as shown below.



This post-release data was collected continuously for three years (2015 through 2017). Though data collection was stopped for all new software projects starting 2018, post-release data for first 12 months was collected for 2017 projects. The abbreviations N and C for each product used in the worksheet is to define the New defects and the Corrected defects found respectively.

The data worksheet contains the following information:

**Rel Date:** It has the release date of each software product.

**Product:** It has the name of each software product that was released.

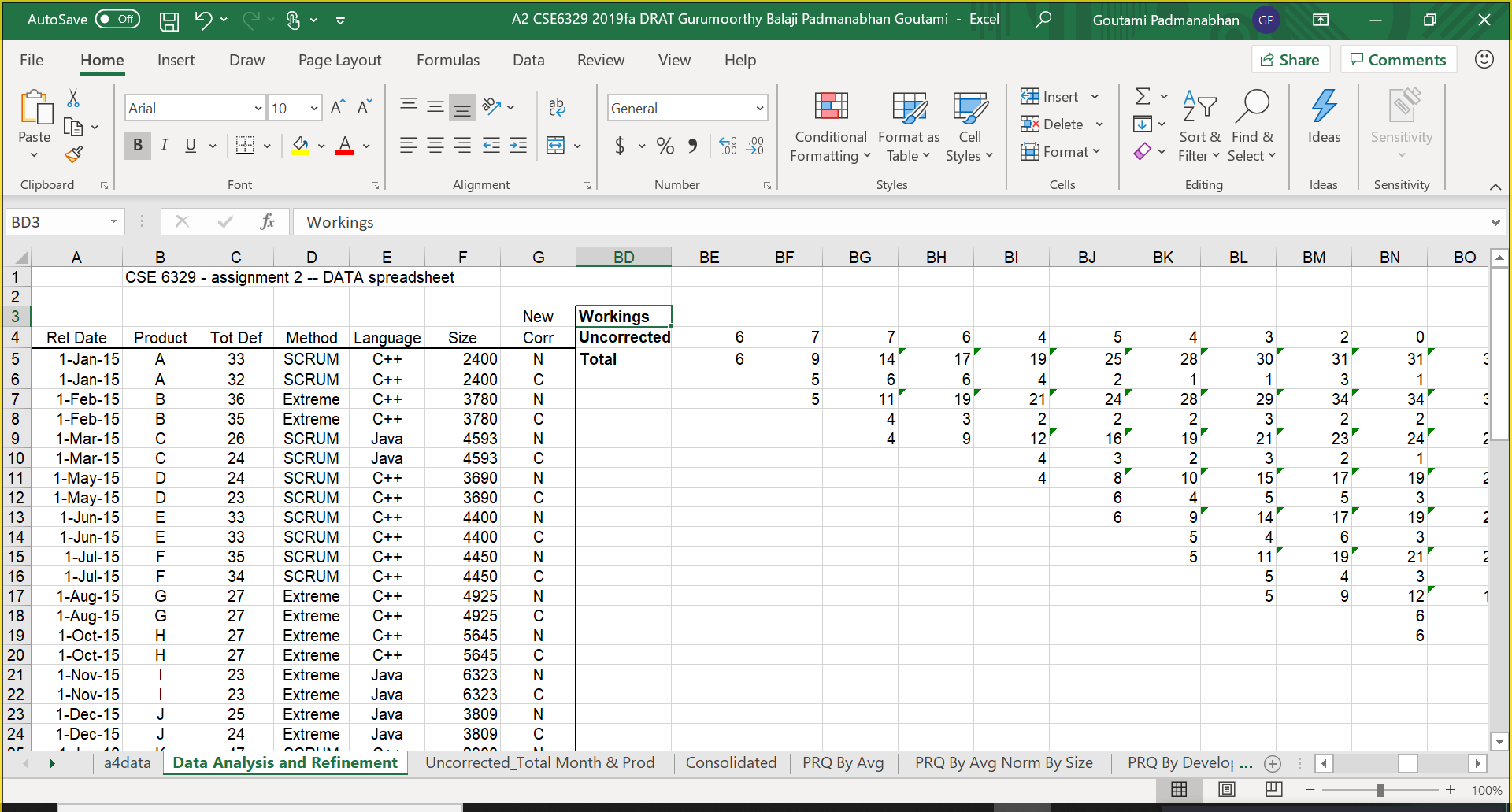
**Tot Def:** It has the total number of new defects found in the software product that was released.

**Method:** It has the development process that was used while implementing the software product namely Scrum and Extreme programming.

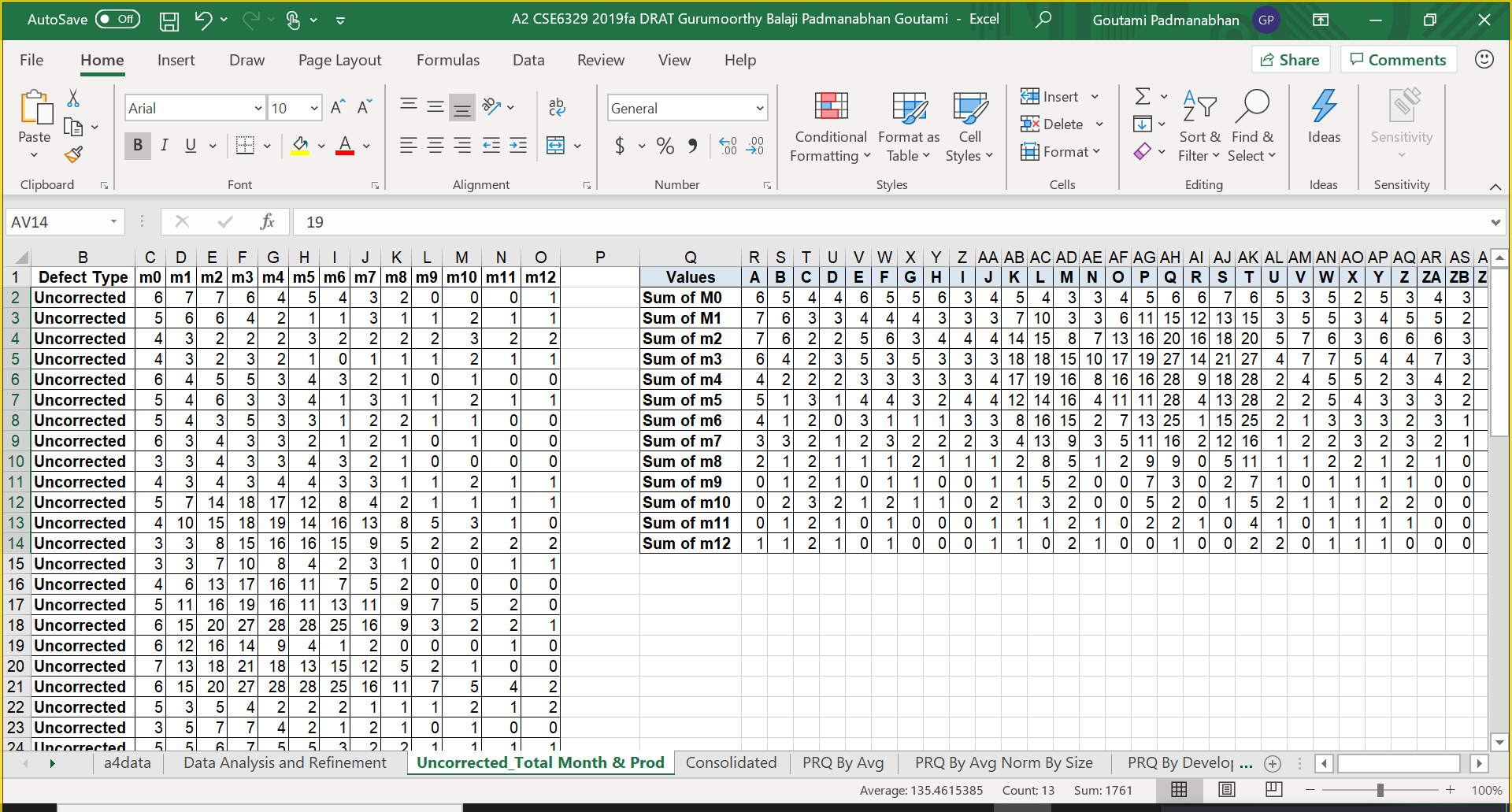
**Language:** It has the name of the programming language that was used to develop the software product such as C++ and Java.

**Size:** It has the number of lines of code present in the software product.

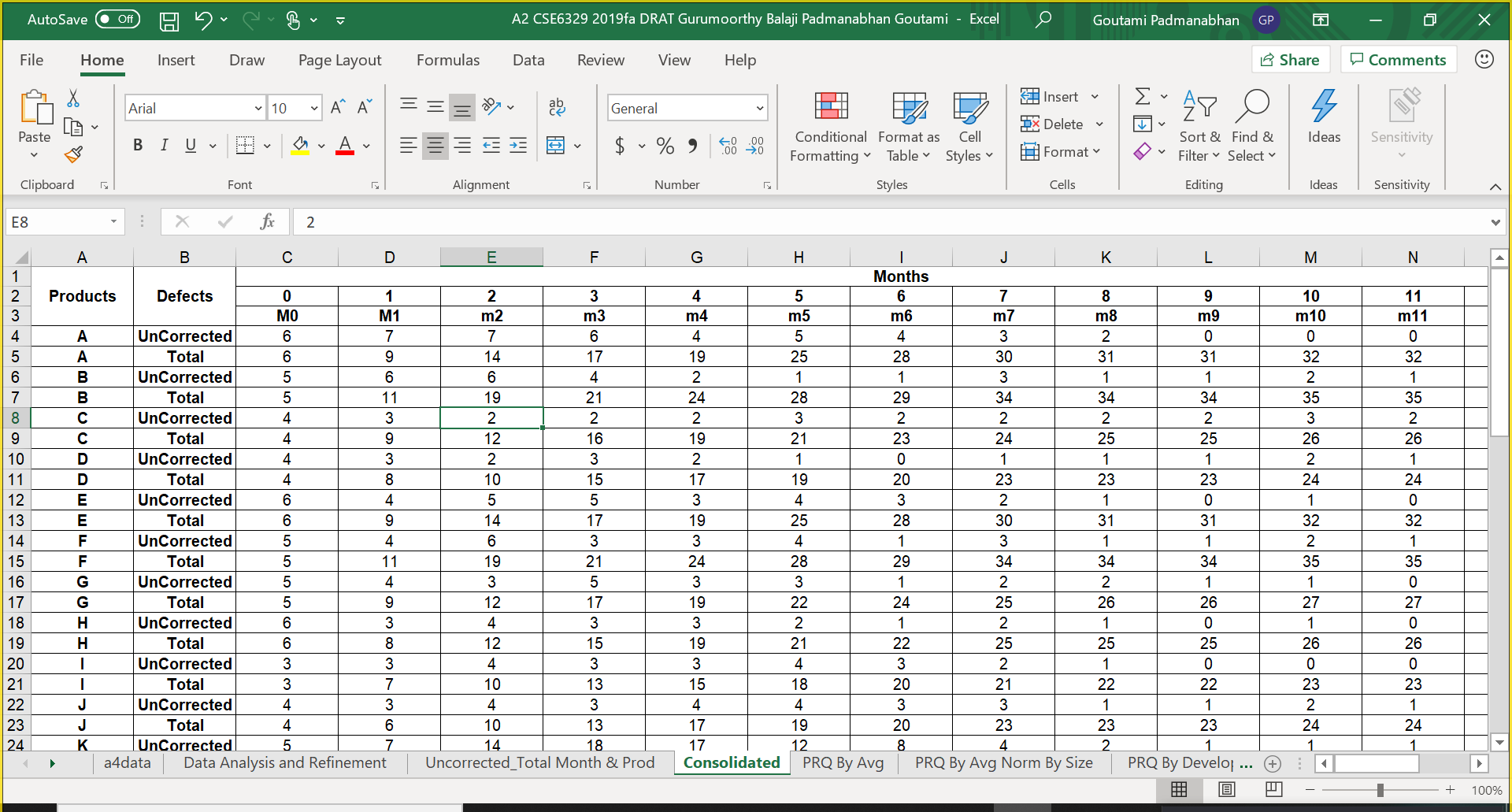
Data analysis and refinement was done on the data by calculating the number of uncorrected defects for each month. The remaining number of new defects were calculated by subtracting the new defects found each month with the corrected defects for each month. The analysis and calculations are shown in the worksheet named ‘Data Analysis and Refinement.xlsx’.



The total defects and the total uncorrected defects for each month are computed separately in a worksheet named ‘Uncorrected\_Total Month & Prod’ as shown below.



The total defects and the total uncorrected defects for each month is listed product-wise in a consolidated sheet named ‘Consolidated.xlsx’ as shown below.



**1.4 Summary of Analyses and Graphs**

The tabular column below lists the Graphs that will be discussed in this report and section of the report where it is discussed along with a short description of its purpose.

|  |  |  |
| --- | --- | --- |
| **Graphs** | **Report section** | **Purpose** |
| Post-Release Quality Average for all products, Normalized by Size | Section 2.1. a. | This graph is derived from the measurements of the number of defects in the software product. It shows how the total defects grow after the release of the product. |
| Post-Release Quality Average for all products, By Development process | Section 2.1. b. | This graph is derived from the measurements of the uncorrected and total defects of Development processes, Scrum and Extreme programming. It shows the differences in the number of defects between two development processes and which process produces more defects than the other. |
| Post-Release Quality Average for all products, By Programming Language | Section 2.1. c. | This graph is derived from the measurements of the uncorrected and total defects of Programming Languages, C++ and Java. It shows the differences in the number of defects between two Programming Languages and which language produces more defects than the other. |
| Post-Release Quality History | Section 2.1. d. | This graph is derived from the measurements of the total new defects overall. It depicts how good the products were after being shipped each month, over several years, in terms of defect rates. Also, it shows how defect rates have changed over time. |
| Current Quality Total | Section 2.1. e. | This graph shows the measurement of total uncorrected defects each month. It is designed to show how many known defects are there each month for all currently active products |
| Current Quality Total, Normalized by Size | Section 2.1. f. | This graph contains the measurement of total uncorrected defects each month with its value normalized by the size. It shows how the total uncorrected defects grow after the release of the product |
| Current Quality Total, By Number of Products | Section 2.1. g. | This graph is derived from the measurements of the uncorrected total defects. It shows the how the number of uncorrected defects have increased or decreased each month. |

1. **Measures, Graphs and Analysis**
   1. **Post Release Quality**

**2.1.a. Post-Release Quality Average, Normalized by Size**

**i. Overview**

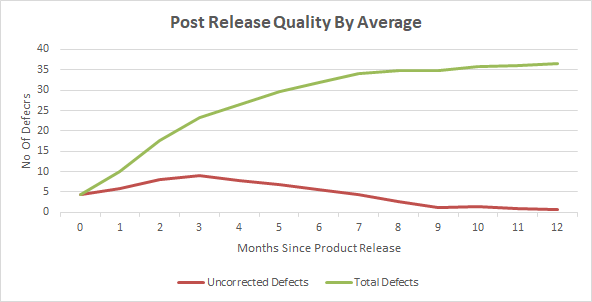
From the data provided there are 30 products released during three-year period from 2015 through 2017. The below graph shows the average for all the products normalized by size i.e. total new defects of each product divided by the product size and multiplied by 1000 to get defects per 1000 lines of code.

The graph is designed as a line chart having two lines, one representing total defects and the other, total uncorrected defects each month after product release for the 12-months. The horizontal axis is the months since release starting at 0 and the vertical axis is defects per 1000 lines of code.

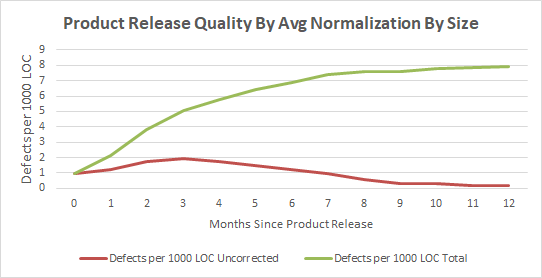
The purpose of this graph is to show the total defects per 1000 lines of code and whether effort has been put to reduce them in order to maintain the quality of product after its release.

**ii. Graph**

The graphs below show Post Release Quality by Average and Post Release Quality by average Normalized by Size for all 30 products. A post release quality by average graph is drawn with “Months Since Product Release” in horizontal axis and “Number of defects” in the vertical axis.



A post release quality by average normalized by size graph is drawn with Months since product release in horizontal axis and Number of defects in the vertical axis.



**iii. Analysis and Discussion**

1. **General Discussion**

The above graphs show that the total defects are increasing every month and the uncorrected defects are reducing every month which shows that the defects are being corrected then and there and focus is mainly on the quality of the software product. From the graph we infer that the product has become stable as more defects are corrected from 7th month. It is observed that the total number of defects are considerably reduced when compared to the initial months post release of the product.

In the second graph the Normalization by size is obtained by dividing the average defects by their corresponding product size and multiplying it with 1000. From the graph it is understood that if one product is larger than the other then it simply means that the it has more number of lines of code than the second one. Normalization eliminates the larger size as a reason for the greater number of defects and enables a better comparison.

Planning helps us to do better comparison of the different products developed using different programming languages regardless of their size. This comparison helps us to improve the quality of the code by reducing the number of defects per 1000 lines of code.

1. What you learned about the various defect/failure rate hypotheses

| Hypothesis | What this graph tells us about this hypothesis. | Summary (-- - 0 + ++) |
| --- | --- | --- |
| 1. There are not any significant failure rate increases. | The graph strongly refutes this as the total defects have kept increasing post release. However, the uncorrected defects were found to be decreasing which shows defects were corrected continuously. | -- |
| 1. Higher defect levels are due to programming language. | This graph doesn’t tell anything about the effect of programming language on defects rate increase/decrease. | 0 |
| 1. Higher defect levels are due to development process. | This graph doesn’t tell anything about the effect of programming language on defects rate increase/decrease. | 0 |
| 1. Defect increases are due to the fact that new products are larger. Defects/LOC are the same, | The graph strongly supports this hypothesis as the size of the product increases the lines of code also increases which results in more defects | ++ |
| 1. Increased defects are due to the fact that we have more products in use. | This graph doesn’t tell anything about the effect of programming language on defects rate increase/decrease. | 0 |
| 1. The quality assurance program in 2017 improved things. | The graph is plotted for every month and also the graph doesn’t talk about quality assurance program. | 0 |

**2.1.b Post Release Quality Average for All Products, by Development Process**

**i. Overview**

This graph shows the Post Release Quality Average for all products based on the number of defects detected in the products that use development process like SCRUM and Extreme Programming.

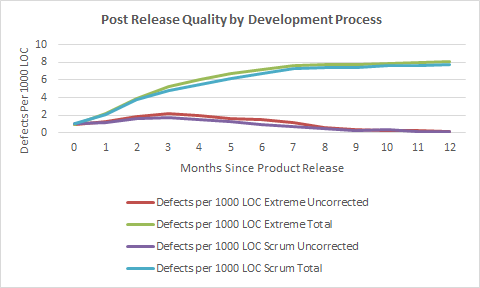
The graph considered is line chart and which displays four lines in it namely total defects & total uncorrected defects per KLOC for SCRUM and Extreme Programming and SCRUM projects respectively.

The graph shows months on the horizontal axis and defects per 1000 LOC on the vertical

axis.

**ii. Graph**

The graph below shows Post Release Quality by Development Process for Extreme Programming and SCRUM processes.



**iii. Analysis and Discussion**

1. **General Discussion**

From the above graph, it is observed that extreme programming had more defects over time especially from 4th month with respect to scrum. Also, the uncorrected defects are more for extreme programming when compared to scrum in the initial period from 4th month till 9th month. But after 9th month it is observed that both extreme programming and scrum processes had more or less the same number of defects (total and uncorrected) per 1000 lines of code.

In this graph the data is obtained from the previous graph data which was normalized by size. Here the uncorrected and total defects are separated from each other based on development process namely extreme programming and scrum. This tabulated data is represented in the form of graph.

1. What you learned about the various defect/failure rate hypotheses

| Hypothesis | What this graph tells us about this hypothesis. | Summary (-- - 0 + ++) |
| --- | --- | --- |
| 1. There are not any significant failure rate increases. | The graph strongly refutes this as the total defects have kept increasing post release. However, the uncorrected defects for both scrum and extreme programming were found to be decreasing which shows defects were corrected/fixed continuously. | -- |
| 1. Higher defect levels are due to programming language. | This graph doesn’t tell anything about the effect of programming language on defects rate increase/decrease. | 0 |
| 1. Higher defect levels are due to development process. | The graph supports this as the total defects using both scrum and extreme programming processes have kept increasing post release. However, the difference between number of defects found using scrum and extreme programming is minimum. | + |
| 1. Defect increases are due to the fact that new products are larger. Defects/LOC are the same, | The graph strongly supports this hypothesis as the size of the product increases the lines of code also increases which results in more defects. | ++ |
| 1. Increased defects are due to the fact that we have more products in use. | This graph doesn’t tell anything about the effect of programming language on defects rate increase/decrease. | 0 |
| 1. The quality assurance program in 2017 improved things. | The graph is plotted for every month and also the graph doesn’t talk about quality assurance program. | 0 |

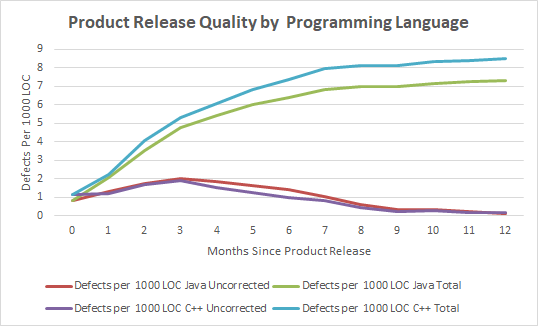
**2.1.c Post Release Quality Average for All Products, by Programming Language**

**i. Overview**

The Post Release Quality Average for all products developed by two programming languages namely Java and C++ were recorded each month for twelve months after a product was released using a line chart. Here it has four lines which are total defects per 1000 lines of code for projects developed using Java, total defects per 1000 lines of code for projects developed using C++, total uncorrected defects for per 1000 lines of code for projects developed using C++ and Java. In the graph below which is designed as a line chart, the horizontal axis represents the months since release starting from 0 and the vertical axis represents the number of defects per 1000 lines of code. This graph answers few of the questions like "Which programming language has resulted in more total defects?” and “Which programming language has got lower uncorrected defects per 1000 LOC?” and also “Which programming language makes it easy to fix defects by debugging?".

**ii. Graph**

The graph below shows Post Release Quality by programming language for C++ and Java respectively.



**iii. Analysis and Discussion**

1. **General Discussion**

We analyzed the defect data and normalized it by the product size per 1000 lines of code like the previous graph and designed this graph.

From the above graph, we can find that the total number of defects for C++ was significantly higher as compared to Java. Despite that fact, the quality of the products developed using C++ got much better than that of Java after 3rd month of release even though the difference between their total number of defects is much significant. Also We can understand that the products developed using C++ produced more defects but those defects were resolved in a much better pace from 3rd month post release which may be due to quick action by the programmers to debug and fix those defects and also may be because of additional resources put in place in order to fix the defects associated with products developed using C++ and also the products developed using C++ are much easier to debug and fix the defects.

1. What you learned about the various defect/failure rate hypotheses

| Hypothesis | What this graph tells us about this hypothesis. | Summary (-- - 0 + ++) |
| --- | --- | --- |
| 1. There are not any significant failure rate increases. | The graph strongly refutes this as the total defects have kept increasing post release. However, the uncorrected defects for both C++ and Java were found to be decreasing which shows defects were corrected/fixed continuously. | -- |
| 1. Higher defect levels are due to programming language. | The graph strongly supports this as the total defects using both C++ and Java programming language have kept increasing post release. | ++ |
| 1. Higher defect levels are due to development process. | This graph doesn’t tell anything about the effect of development process on defects rate increase/decrease. | 0 |
| 1. Defect increases are due to the fact that new products are larger. Defects/LOC are the same, | The graph strongly supports this hypothesis as the size of the product increases the lines of code also increases which results in more defects. | ++ |
| 1. Increased defects are due to the fact that we have more products in use. | This graph doesn’t tell anything about the effect of programming language on defects rate increase/decrease. | 0 |
| 1. The quality assurance program in 2017 improved things. | The graph is plotted for every month and also the graph doesn’t talk about quality assurance program. | 0 |

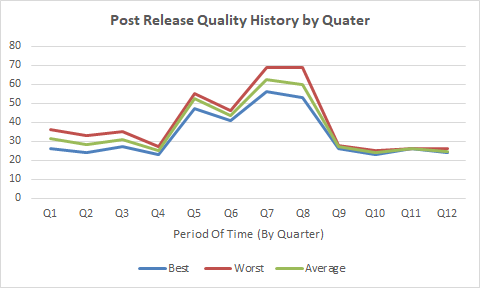
**2.1.d Post Release Quality History by Quarter**

**i. Overview**

The Post Release Quality History by Quarter shows the level of quality of product post release by quarter. This is said to be a “lagging” indicator, as it can only be computed after a product release which is out for 12 months. The graph shows the total number of defects found for all products in that respective quarters during their first 12 months of use.

The below graph is designed as a line chart and by taking into account the best case, worst case, average case values of defects of products for every quarter i.e., the number of the defects for all the products that were released during that quarter. The horizontal axis represents a period of time by quarter and the data on the horizontal axis represents all products released during that period of time (quarter) and vertical axis represents post release quality for all products released during that time period.

**ii. Graph**



**iii. Analysis and Discussion**

1. **General Discussion**

From the graph shown above, we can analyze that the number of defects of products for every quarter by taking into account the best case, worst case, average case values it is clear that the defects count has drastically increased after quarter 4 till quarter 8 post release of the products and all the three lines in the graph i.e. best case, worst case and average case of defects has almost very marginal difference between them as shown in the above graph.

1. What you learned about the various defect/failure rate hypotheses

| Hypothesis | What this graph tells us about this hypothesis. | Summary (-- - 0 + ++) |
| --- | --- | --- |
| 1. There are not any significant failure rate increases. | The graph refutes this as the total defects have kept increasing post release every quarter from quarter 4 till quarter 8. However, the defects were found to be decreasing after Quarter 8 which shows defects were corrected/fixed. | - |
| 1. Higher defect levels are due to programming language. | This graph doesn’t tell anything about the effect of programming language on defects rate increase/decrease. | 0 |
| 1. Higher defect levels are due to development process. | This graph doesn’t tell anything about the effect of development process on defects rate increase/decrease. | 0 |
| 1. Defect increases are due to the fact that new products are larger. Defects/LOC are the same, | This graph doesn’t tell anything about the effect of LOC or new products size. | 0 |
| 1. Increased defects are due to the fact that we have more products in use. | The graph refutes this as the number of products released every four quarters are the same which shows that the number of products in use has no effect on increase in defects. | - |
| 1. The quality assurance program in 2017 improved things. | The graph strongly supports the hypothesis as the defects have got reduced drastically in the last four quarters i.e., during the year 2017. | ++ |

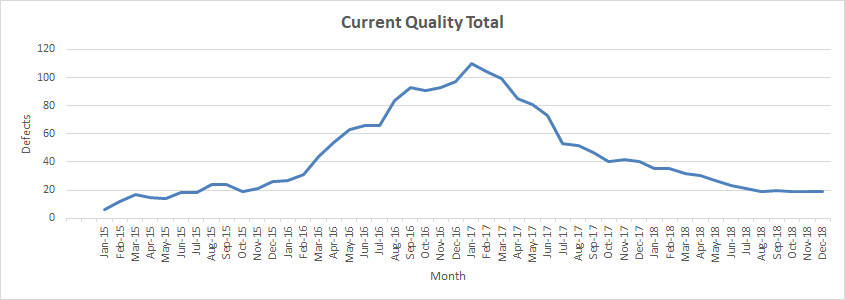
* 1. **Current Quality**

**2.2.a. Current Quality Total**

**i. Overview**

Current quality total helps us to understand the present quality of the product based on the total uncorrected defects and it is an important metric to measure how much additional resources we may need to assign for correcting defects and in-order to bring down the uncorrected defects to a lower value. The Current Quality Total is measured for all active products for each month and is defined as the total uncorrected defects for all active products. We have considered line chart to depict this graph.

**ii. Graph**



**iii. Analysis and Discussion**

1. **General Discussion**

As the current Quality Total talks about the total uncorrected defects for all active products, from the above graph we interpret that the total uncorrected defects in the first few months of the Year 2015 was less and then it has increased considerably from January 2016 till December 2016. Then the same has started to decrease from January 2017 and got gradually reduced till December 2018. This is because the personnel have started working more on uncorrected defects and brought it down from the beginning of the year 2017 when compared to previous year 2016. Finally, the values have become constant from August 2018 till the end of that year.

1. What you learned about the various defect/failure rate hypotheses

| Hypothesis | What this graph tells us about this hypothesis. | Summary (-- - 0 + ++) |
| --- | --- | --- |
| 1. There are not any significant failure rate increases. | The graph strongly refutes this as the total uncorrected defects have kept increasing every month post release. However, the defects were found to be decreasing from the beginning of 2017 which shows defects were corrected/fixed. | -- |
| 1. Higher defect levels are due to programming language. | This graph doesn’t tell anything about the effect of programming language on defects rate increase/decrease. | 0 |
| 1. Higher defect levels are due to development process. | This graph doesn’t tell anything about the effect of development process on defects rate increase/decrease. | 0 |
| 1. Defect increases are due to the fact that new products are larger. Defects/LOC are the same, | This graph doesn’t tell anything about the effect of LOC or new products size. | 0 |
| 1. Increased defects are due to the fact that we have more products in use. | This graph doesn’t tell anything about the effect of number of products in use on defects. | 0 |
| 1. The quality assurance program in 2017 improved things. | The graph supports the hypothesis for the period 2015 through 2017 but from 2018 even though the QA program was stopped the uncorrected defects have become constant. | + |

**2.2.b. Current Quality Total Normalized by Size**

**i. Overview**

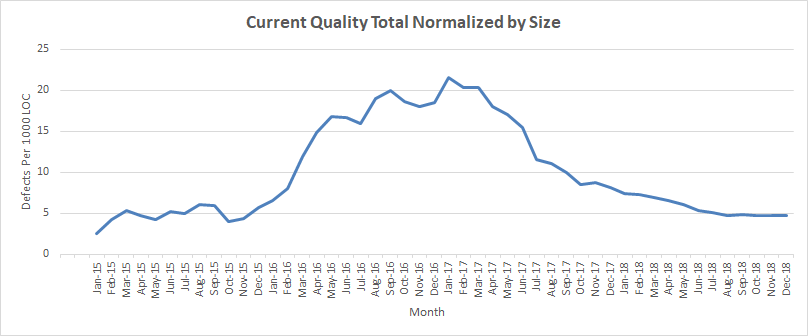
From the data provided we have designed the below graph and it shows the current quality total for all the active products normalized by size i.e. total uncorrected defects of each product divided by its respective product size and multiplied by 1000 to get defects per 1000 lines of code.

The graph is designed as a line chart where the horizontal axis is the months since release starting at January 2015 till Dec 2018 and the vertical axis is defects per 1000 lines of code with respect to total uncorrected defects.

The purpose of this graph is to show the total uncorrected defects per 1000 lines of code based on their product size and it also shows whether proper effort has been put by the programmer to bring down the uncorrected defect count normalized by size to a value lower in order to maintain the quality of product after its release. The current quality total normalized by size, for each month, for 3 years, using all active products each month is shown below.

**ii. Graph**

The Current Quality Total Normalized by Size graph has month in horizontal axis and defects per 1000 lines of code in vertical axis depicting the total uncorrected defects normalized by size from the year 2015 till 2018 for all active products.



**iii. Analysis and Discussion**

1. **General Discussion**

The Total Uncorrected defects for all the active products in the year 2016 and 2017 was way higher than in 2015 or 2018 when normalized by size which means taking into consideration defects per 1000 lines of code per product size. The Uncorrected defects is decreasing from the middle of the year 2017 from June 2017 and gradually decreases and becomes constant at the end of the year 2018. This means that products from K thru Y had the highest uncorrected defects in that time period. The highest number of defects were in January 2017.

1. What you learned about the various defect/failure rate hypotheses

| Hypothesis | What this graph tells us about this hypothesis. | Summary (-- - 0 + ++) |
| --- | --- | --- |
| 1. There are not any significant failure rate increases. | The graph strongly refutes this as the total uncorrected defects have kept increasing every month post release till Feb 2017. However, the defects were found to be decreasing after Feb 2017 which shows defects were corrected/fixed. | -- |
| 1. Higher defect levels are due to programming language. | This graph doesn’t tell anything about the effect of programming language on defects rate increase/decrease. | 0 |
| 1. Higher defect levels are due to development process. | This graph doesn’t tell anything about the effect of development process on defects rate increase/decrease. | 0 |
| 1. Defect increases are due to the fact that new products are larger. Defects/LOC are the same, | This graph supports the hypothesis but not completely as the products got larger especially from Jan 2016 till Jan 2017 which resulted in more LOC which in-turn resulted in more defects. | + |
| 1. Increased defects are due to the fact that we have more products in use. | This graph doesn’t tell anything about the effect of number of products in use on defects. | 0 |
| 1. The quality assurance program in 2017 improved things. | The graph supports the hypothesis for the period 2015 through 2017 but from 2018 the uncorrected defects have become constant even though the QA program was stopped during that period. | + |

**2.2.c. Current Quality Total Normalized by Number of Products**

**i. Overview**

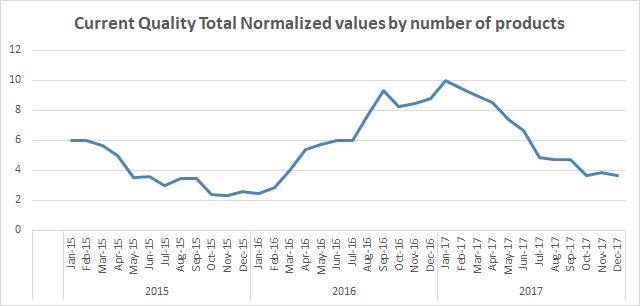
From the data provided we have designed the below graph and it shows the current quality total normalized by number of products i.e. total uncorrected defects of that month divided by the no of active products of that month.

The graph is designed as a line chart where the horizontal axis is the months since release starting at January 2015 till Dec 2017 and the vertical axis is normalized value of uncorrected defects.

The purpose of this graph is to show the total uncorrected defects normalized by number of active products as of that month based on their product size and it also shows whether proper effort has been put to reduce the uncorrected defects in order to maintain the quality of product after its release. The current quality total normalized by number of products, for each month, for 3 years, using all active products each month is shown below.

**ii. Graph**

The Current Quality Total Normalized by Number of Products graph has month in horizontal axis and defects (uncorrected) per no of products released in vertical axis depicting the total uncorrected defects normalized by number of products from the year 2015 till 2017 for all active products.



**iii. Analysis and Discussion**

1. **General Discussion**

The Total Uncorrected defects got increased from January 2016 till Jan 2017 with a slight dip in value in Oct, Nov and Dec 2016 and then started to decrease gradually from February 2017 till the end of the year 2017.

1. What you learned about the various defect/failure rate hypotheses

| Hypothesis | What this graph tells us about this hypothesis. | Summary (-- - 0 + ++) |
| --- | --- | --- |
| 1. There are not any significant failure rate increases. | The graph refutes this as the total uncorrected defects normalized by number of products has seen a decrease from Jan 2015 till Dec 2015 and again increased from Jan 2016 to Dec 2016 and then reduced again later. | - |
| 1. Higher defect levels are due to programming language. | This graph doesn’t tell anything about the effect of programming language on defects rate increase/decrease. | 0 |
| 1. Higher defect levels are due to development process. | This graph doesn’t tell anything about the effect of development process on defects rate increase/decrease. | 0 |
| 1. Defect increases are due to the fact that new products are larger. Defects/LOC are the same, | This graph doesn’t tell anything about the effect of number of products in use on defects. | 0 |
| 1. Increased defects are due to the fact that we have more products in use. | This graph refutes this hypothesis as the number of active products became constant from Dec 2015 till Dec 2017 as there were not more products in use. | - |
| 1. The quality assurance program in 2017 improved things. | The graph strongly supports the hypothesis as the defects have got reduced gradually from the start of the year 2017. | ++ |

**3.0 Summary and Recommendations**

**3.1 Summary**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Hypothesis** | **Graphs / Measures** | | | | | | | **Overall** | **Comments** |
|  | **1a** | **1b** | **1c** | **1d** | **2a** | **2b** | **2c** |  |  |
| **1 no issue** | -- | -- | -- | - | -- | -- | - | -- | There is a significant increase in number of defects. |
| **2 language** | 0 | 0 | ++ | 0 | 0 | 0 | 0 | 0 | Language doesn’t seem to be an issue with the number of defects as the graphs tell little or nothing about the effect of programming language on defects. |
| **3 process** | 0 | + | 0 | 0 | 0 | 0 | 0 | 0 | Use of one development process over the other does not benefit in reducing the number of defects. Both the processes show more or less the same effect on defect control. |
| **4 size** | ++ | ++ | ++ | 0 | 0 | + | 0 | ++ | The Product Size is a major factor as when product size increases the lines of code also increases there by increasing the probability of more defects which is a concern. Further analysis can be done on various aspects of product size. |
| **5 Number of Products** | 0 | 0 | 0 | - | 0 | 0 | - | 0 | The number of products in use have not increased or decreased year on year and hence we are inconclusive about the hypothesis. |
| **6 QA program** | 0 | 0 | 0 | ++ | + | + | ++ | + | From the data analyzed, we found that the QA Program may have little bit of contribution in reducing the number of defects. Further analysis could be done with data on number of people who attended QA Programs and involved in fixing defects and improving the quality of various products. |

**3.2 Recommendations**

In addition to the recommendations stated above in summary table, we recommend that the size of the software product could be reduced wherever possible in-order to keep the number of lines of code minimum. One way to do this is by using C++ as primary [programming language for development of software products as it has less lines of code when compared to Java. We can also try introducing other efficient programming languages such as Python which can reduce lines of code at least by 20 -30% which results in reduced product size and number of defects there by increasing product quality.

We also recommend SCRUM as the process to be followed for software development as the number of defects found were less for SCRUM when compared to extreme programming.

Also, we recommend reinstating the QA program as its effect can be seen from the data especially during 2015 & 2017 where the defects have been kept minimal and reduced drastically. A dedicated QA team for fixing defects should be allotted in-order to keep the defect rate under control.