SOEN 6011
Software Engineering Processes

Project Report
Function 6 : $B(x,y)$ Beta Function

Supervised By:
Dr. Pankaj Kamthan

Mahavir Nareshbhai Patel (40198619)

August 05, 2022

1 Problem 1

1.1 Introduction

The beta function is a special function that belongs to the first category of Euler's integrals. The beta function is denoted by the symbol "B". $B(x,y)$ refers to the beta function, where x and y are real-valued parameters.
The formula of $B(x,y)$ is:

- $B(x,y) = \frac{(x-1)!(y-1)!}{(x+y-1)!}$ For positive integers[1]
- $B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}dt$ For positive real numbers

1.2 Properties of Beta Function

- Beta function is symmetric which means its beta value is independent of the order of its parameters: $B(x,y) = B(y,x)$
- $B(x,y) = B(x,y+1) + B(x+1,y)$
- $B(x,y+1) = B(x,y) \cdot [\frac{y}{x+y}]$
- $B(x+1,y) = B(x,y) \cdot [\frac{x}{x+y}]$
- $B(x,y).B(x+y,1-y) = \frac{\pi}{x}\sin\pi y$
- Beta function in terms of Gamma functions as: $B(x,y) = \frac{\Gamma x \Gamma y}{\Gamma(x+y)}$, When x and y are positive whole number then it follows the form of gamma function.
- The beta function can be extended to a function with more than two arguments: $B(X_1,X_2,...X_n) = \frac{\Gamma X_1 \Gamma X_2 ... \Gamma X_n}{\Gamma(X_1+X_2+...+X_n)}$

1.3 Domain and Co-Domain

The domains of real numbers are where the beta function is defined. The limitations of the integral function determine the beta function's co-domain. the beta function is defined For real values that are positive and greater than zero. nonetheless, there are other ways to write a beta function, for example,

- $B(x,y) = \int_0^{\frac{\pi}{2}} (\sin\theta)^{2x-1}(\cos\theta)^{2x-1}$ where $x > 0$ and $y > 0$
- $B(x,y) = \int_0^{\infty} \frac{t^{x-1}}{(1+t)^{x+y}}dt$ where $x > 0$ and $y > 0$

for the variables domain is (0,∞] and based on the beta function's integral limits co-domain can be defined.

2 Context of Use Model

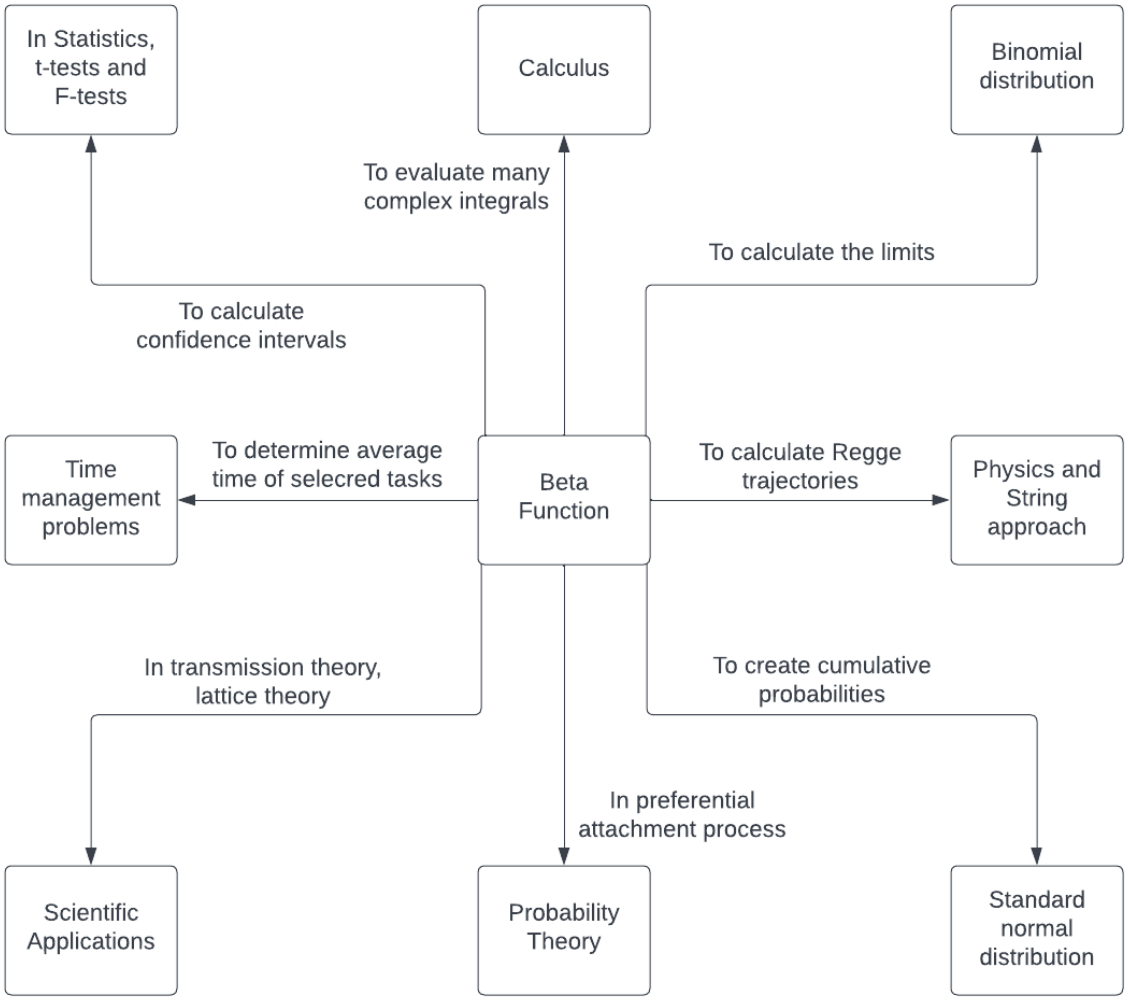

Figure 2.1: Context of use model

3 Problem 2

Assumptions

1. x and y are positive real numbers $x, y \in R^+$.
2. It is simpler to calculate $B(x,y)$ using the factorial for $x, y \in Z^+$.
3. If x and y are real numbers, there is no need to compute the integral function. The gamma values of the numbers may be used to derive the beta value using Stirling's approach.

1. **First Requirement**
   - **ID =** FR1
   - **Type =** Functional Requirements
   - **Version =** 1.0
   - **Difficulty =** Easy
   - **Description =** The Beta function $B(x,y)$ requires x and y as its two variable inputs in order to operate.
   - **Rationalle =** x and y
2. **Second Requirement**
   - **ID =** FR2
   - **Type =** Functional Requirements
   - **Version =** 1.0
   - **Difficulty =** Easy
   - **Description =** The Beta function $B(x,y)$ requires two real positive numbers as it's defined in the R^+ domain.
   - **Rationalle =** $x \geq 0$ and $y \geq 0$
3. **Third Requirement**
   - **ID =** FR3
   - **Type =** Functional Requirements
   - **Version =** 1.0
   - **Difficulty =** Easy
   - **Description =** The Beta Value of the function is in real positive numbers i.e R^+
   - **Rationalle =** $B(x,y) \geq 0$

4. **Fourth Requirement**
    - **ID =** FR4
    - **Type =** Functional Requirements
    - **Version =** 1.0
    - **Difficulty =** Easy
    - **Description =** If the given inputs are positive integers then beta Values can be easily computed by using the Beta - Gamma Function relation.
    - **Rationalle =** $\{ \forall x, y \in Z^{+} \mid B(x,y) = \frac{\Gamma x \Gamma y}{\Gamma(x+y)} \}$
5. **Fifth Requirement**
    - **ID =** FR5
    - **Type =** Functional Requirements
    - **Version =** 1.0
    - **Difficulty =** Moderate
    - **Description =** To calculate Beta function for large integer values, Gamma Function should be used in order to prevent stack overflow by using tail recursive function.
    - **Rationalle =** $\{ \forall x, y \in R^{+} \mid B(x,y) = \frac{\Gamma x \Gamma y}{\Gamma(x+y)}$ *where* $\Gamma n = (n-1)! \}$
6. **Sixth Requirement**
    - **ID =** FR6
    - **Type =** Functional Requirements
    - **Version =** 1.0
    - **Difficulty =** Difficult
    - **Description =** For the decimal number gamma value can be calculated using the stirlings's approximation which helps in determining the Beta value without using the integral functions.
    - **Rationalle =** $\{ \forall x, y \in R^{+} \mid B(x,y) = \frac{\Gamma x \Gamma y}{\Gamma(x+y)}$ *where* $\Gamma n = \sqrt{2 \cdot \pi \cdot n} \cdot (\frac{n}{e})^{n} \}$
7. **Seventh Requirement**
    - **ID =** FR7
    - **Type =** Functional Requirements
    - **Version =** 1.0
    - **Difficulty =** Moderate
    - **Description =** There is no definition of beta values for negative or zero values. There shouldn't be any inputs besides the numeric values; x and y can be similar or different, but there shouldn't be any inputs other than the numeric values.
    - **Rationalle =** $x \leq 0$ *and* $y \leq 0$, $x = y$ *or* $x \neq y$ *where* $x, y \in R^{+}$

8. **Eighth Requirement**
    - **ID =** QR1
    - **Type =** Non-Functional Requirement
    - **Version =** 1.0
    - **Difficulty =** Easy
    - **Description =** The system should maintainable and changes can be easily applied to the system.
    - **Rationalle =** The maintainability is provided by dividing distinct tasks into different modules or functions.
9. **Ninth Requirement**
    - **ID =** QR2
    - **Type =** Non-Functional Requirement
    - **Version =** 1.0
    - **Difficulty =** Easy
    - **Description =** The system should be portable and able to run on different system such as various operating system.
    - **Rationalle =** The Java (requires version 1.8 or above) programs are compatible with any system architecture that has a JVM (Java Virtual Machine).
10. **Tenth Requirement**
    - **ID =** QR3
    - **Type =** Non-Functional Requirement
    - **Version =** 1.0
    - **Difficulty =** Easy
    - **Description =** The system design should be easy to understand for any user and can easily interpret the error message displayed on the system even for non-technical users.
    - **Rationalle =** The error message should clearly state what went wrong.

4 Problem 3

4.1 Algorithm 1 - Tail Recursive Factorial Function

The gamma function is utilised by the beta function to calculate positive integers. The factorial of the number makes it simple to determine the Gamma Value of a positive integer.

$$B(x,y) = \frac{\Gamma x \Gamma y}{\Gamma(x+y)} \text{ where } \Gamma x = (x-1)!$$

4.1.1 Advantages

- The algorithm Gives the accurate answers for the positive integers.
- The execution speed of the tail recursive function compared to recursive function is very fast and memory efficient.
- This algorithm makes it simple to calculate the integer's beta values.

4.1.2 disadvantages

- Can be used for only Positive numbers.
- Only calculate the Beta Values for the integer
- It does not Consider decimal numbers.

4.1.3 Why Tail Recursive Function ?

If a function concludes by returning the result of the recursive call, it is considered tail-recursive. It is a waste of memory to keep the caller's frame on the stack after the recursive call returns its value because nothing else has to be done. Therefore, the current frame can be used for the call rather than allocating a new one. And for the small value of the integer sometimes recursive factorial faction can cause StackOverFlow error that's why it's better to use the tail recursive function.

4.1.4 Algorithm

Algorithm 1 Calculate Beta Function using factorial

Require: value: $x > 0$ & $y > 0$ ▷ where $x, y \in Z^+$
Ensure: $result = Beta(x, y)$

procedure BETAFUNCTION(x, y)
    $value1 \leftarrow$ CALCULATEGAMMA(x)
    $value2 \leftarrow$ CALCULATEGAMMA(y)
    $value3 \leftarrow$ CALCULATEGAMMA($x + y$)
    $betaValue \leftarrow \frac{value1 * value2}{value3}$
    return $betaValue$ ▷ It returns the beta value
end procedure

procedure GAMMAFUNCTION($value$)
    $value \leftarrow value - 1$
    return FACTORIAL($value$) ▷ It returns the gamma value
end procedure

procedure FACTORIAL($value$)
    return FACTORIALTAILRECURSIVE($value, 1$) ▷ It returns the factorial of the number
end procedure

procedure FACTORIALTAILRECURSIVE($value, n$)
    if $value = 0$ **then**
        return n ▷ Return factorial of the number
    else
        return FACTORIALTAILRECURSIVE($value - 1$, $value * n$) ▷ tail recursive call to function
    end if
end procedure

$result \leftarrow$ CALCULATEBETA(x, y) ▷ Final result of $Beta(x, y)$

4.2 Algorithm 2 - Stirling's approximation

The gamma Stirling's approximation, also referred to as Stirling's formula is a mathematical approximation for factorials for decimal values and can be used to construct the beta function for decimal numbers. Since it is a good approximation, it delivers correct results even for small values of n.

$$B(x,y) = \frac{\Gamma x \Gamma y}{\Gamma(x+y)} \text{ where, } \Gamma n = \sqrt{2\pi n} \cdot (\frac{n}{e})^n$$

4.2.1 Advantages

- The algorithm Gives the answers for the positive real numbers.
- Most values that are available can be computed by the algorithm.
- It is possible to apply this approximation approach in place of the integration function to reduce the complexity of the code

4.2.2 disadvantages

- The algorithm is unable to produce reliable answers.
- Although the complexity is decreased, the earlier algorithm is still substantially more difficult.
- The code cannot be easily debugged.
- The mismatch between the necessary and actual answers is much different for smaller values. However, the difference gets smaller as the size of the numbers grows.

4.2.3 Why to use Stirling's Approximation ?

To calculate the Beta value of the decimal number, it's hard to compute using the gamma's integral function. However, Stirlings' approximation can be applied to reduce the complexity. Even for small numbers, this is a good approximation procedure that yields accurate answers. The best outcome is always obtained through approximation. The comparison of Stirling's approximation and the factorial is provided below.

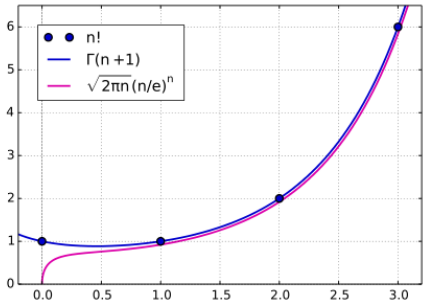

Figure 4.1: Comparison of Stirling's approximation and the factorial. Source:- WikiPedia

4.2.4 Algorithm

Algorithm 2 Calculate Beta Function using Stirlings's approximation

Require: value: $x > 0$ & $y > 0$ ▷ where $x, y \in R^+$
Ensure: $result = Beta(x, y)$

procedure CALCULATEBETA(x, y)
    $value1 \leftarrow$ CALCULATEGAMMA(x)
    $value2 \leftarrow$ CALCULATEGAMMA(y)
    $value3 \leftarrow$ CALCULATEGAMMA($x + y$)
    $beta \leftarrow \frac{value1 * value2}{value3}$
    return $beta$ ▷ It returns the beta value
end procedure

procedure CALCULATEGAMMA($value$)
    $firstPart \leftarrow 2 \cdot \pi \cdot value$
    $secondPart \leftarrow (\frac{value}{e})$
    $gamma \leftarrow$ CALCULATEPOWER($firstPart, \frac{1}{2}$)CALCULATEPOWER($secondPart, value$)
    return $gamma$ ▷ It returns the gamma value
end procedure

procedure CALCULATEPOWER($value1$,$value2$)
    $power \leftarrow math.power(value1, value2)$
    return $power$ ▷ It returns the base to the power
end procedure

procedure CALCULATELOG($value$)
    $answer \leftarrow 0$
    $base \leftarrow \frac{value-1}{value+1}$
    for $i \leftarrow 1, 125$ **do**
        $exponent \leftarrow 2 * i - 1$
        $answer \leftarrow answer + \frac{1}{exponent} *$ CALCULATEPOWER($base, exponent$)
    end for
    return $2 * answer$ ▷ It returns the value of Log n
end procedure

$result \leftarrow$ CALCULATEBETA(x, y) ▷ Final result of $Beta(x, y)$

Algorithm 3 Calculate the power(x,y)

Require: value: x & y ▷ where $x, y \in R$

Ensure: $result = Power(x, y)$

procedure CALCULATEPOWER(*base*,*exponent*)
    Convert the exponent into String
    $exponentArray \leftarrow Split\ the\ exponent\ into\ integer\ and\ fractional\ part$
    if $exponetArray[1] > 0$ **then**
        return CALCULATEFRACTIONPOWER(*base*,*exponent*)
    end if
    if $exponent < 0$ **then**
        $base \leftarrow \frac{1}{base}$
        $exponent \leftarrow (-1) \cdot exponent$
    end if
    if $exponent \leftarrow 0$ **then**
        return 1
    end if
    if $exponent\%2 \leftarrow 0$ **then**
        $base \leftarrow base * base$
        $exponent \leftarrow \frac{exponent}{2}$
        return CALCULATEPOWER(*base*,*exponent*)
    else
        $exponent \leftarrow \frac{exponent-1}{2}$
        return *base* * CALCULATEPOWER$(base * base, exponent)$
    end if
end procedure

procedure CALCULATEFRACTIONPOWER(*base*,*exponent*)
    $answer \leftarrow 0$
    $logvalue \leftarrow 0$
    if $exponet \leftarrow 0$ **then**
        return 1
    end if
    if $base < 0$ **then**
        $logvalue \leftarrow$ CALCULATELOG$(base * (-1))$
    else
        $logvalue \leftarrow$ CALCULATELOG$(base)$
    end if
    if $exponent \leftarrow 0 \wedge exponent > 0$ **then**
        return *answer*
    end if
    for $i \leftarrow 0, 125$ **do**
        $numerator \leftarrow$ CALCULATEPOWER$(exponent * logvalue, i)$
        $denominator \leftarrow$ FACTORIAL(i)
        $answer \leftarrow answer + \frac{numerator}{denominator}$
    end for
    if $base < 0 \wedge\ exponent\%2 \neq 0$ **then**
        return $answer * (-1)$
    else
        return *answer*
    end if
end procedure

4.3 Mind Map

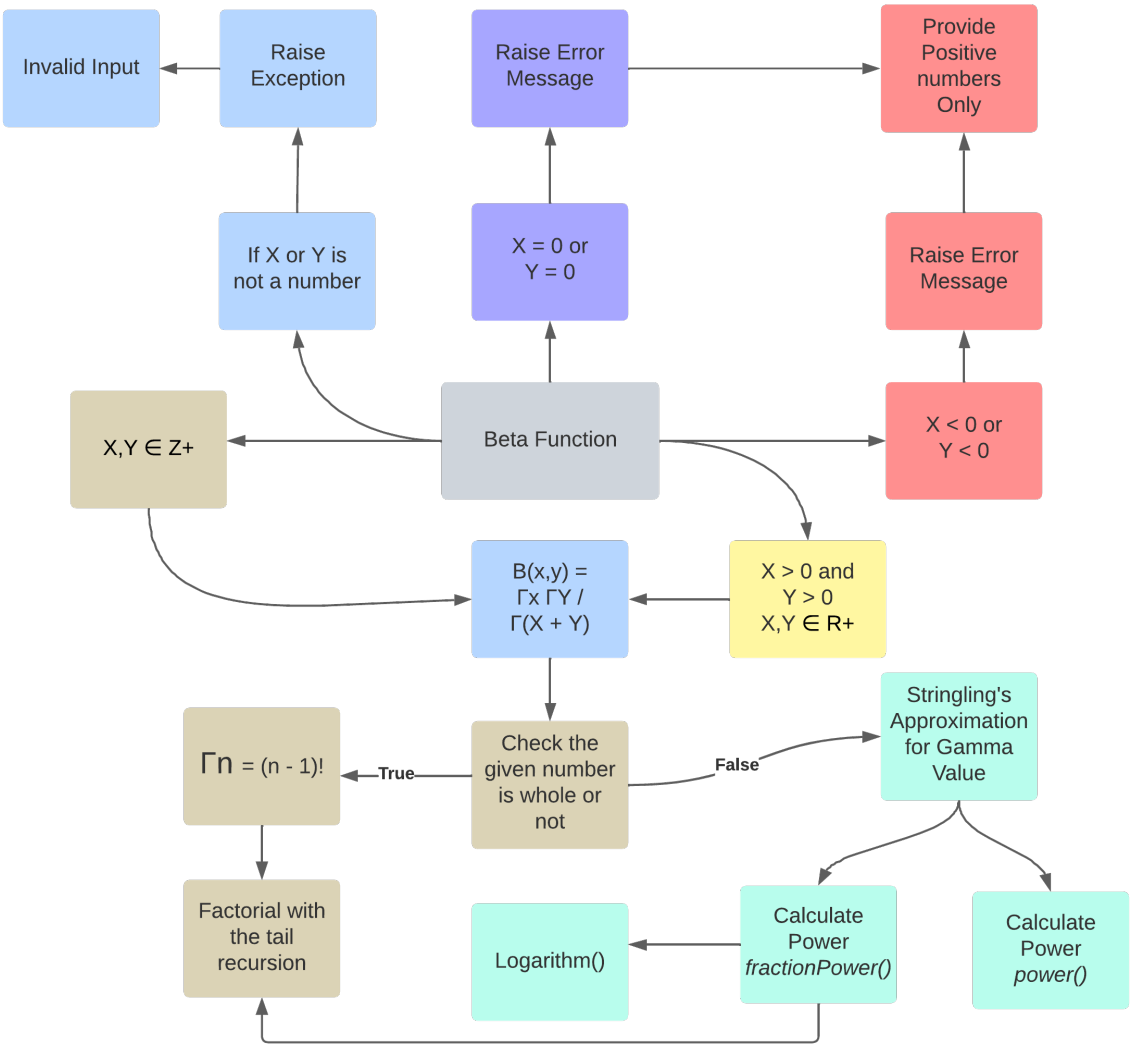

Figure 4.2: Mind Map for Pseudo Code

5 Problem 4

5.1 Code Review for Function-6 Beta Function $B(x,y)$

5.1.1 Debugger

A debugger is computer software or a tool that programmers use to test and debug a target program. By employing instruction-set simulators rather than running a program directly on the CPU, debuggers can exert more control over how it is executed. This makes it possible for debuggers to pause or terminate the application in response to specific events. In this case, the debugger tool of the Eclipse IDE is used to troubleshoot the code of the beta function calculator. Breakpoints are utilized to temporarily halt the execution of the code.

Advantages

- Able to examine and analyse the values of a stack, heap, or variable at a single line of code.
- Can stop the program's execution at a specific moment to examine its path and its values with the help breakpoints.
- Can be easily applied to the running program.

Disadvantages

- Does not support real time and multi threading ahdoc
- May not able to expose all the problems

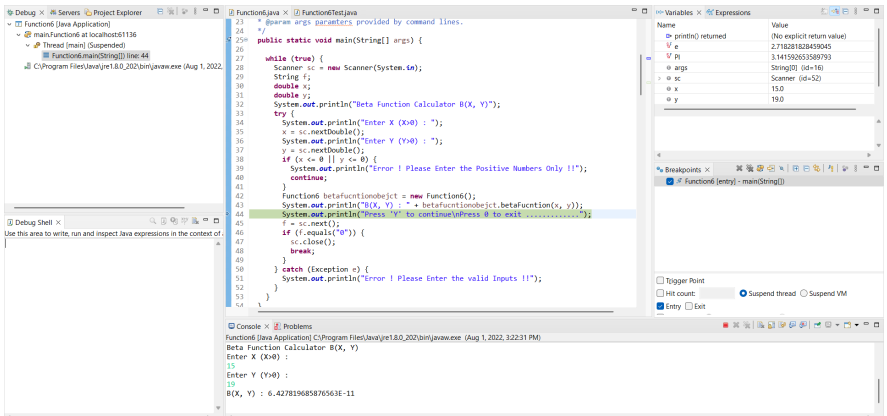

Figure 5.1: Snapshot of the Eclipse IDE debugger tool

5.2 CheckStyle Tool

The ”CheckStyle” plugin for Eclipse provides the code specifications and guarantees that the java code adheres to the accepted code styles. Java code for the beta function was validated using the Google check style. Following snapshot shows there is not any coding standard violations in the Beta Function code.

Advantages

- Supports Automatic Checking and can be used in the ongoing project. automatically checks the coding standards line by line.
- Supports multiple types of check styles such as Google and Sun microsystems coding standards.
- CheckStyle tool is portable and can be used with any IDEs and help to unify the team across various boundaries.

Disadvantages

- To validate the coding standard, it needs to be re-compiled every time
- Only validate the code specification rather than apply the changes and modify them according to the given checkstyle specification.
- This tool has its flaws. needs to be refreshed or sometimes have to clean and build the project again to refrain from errors.

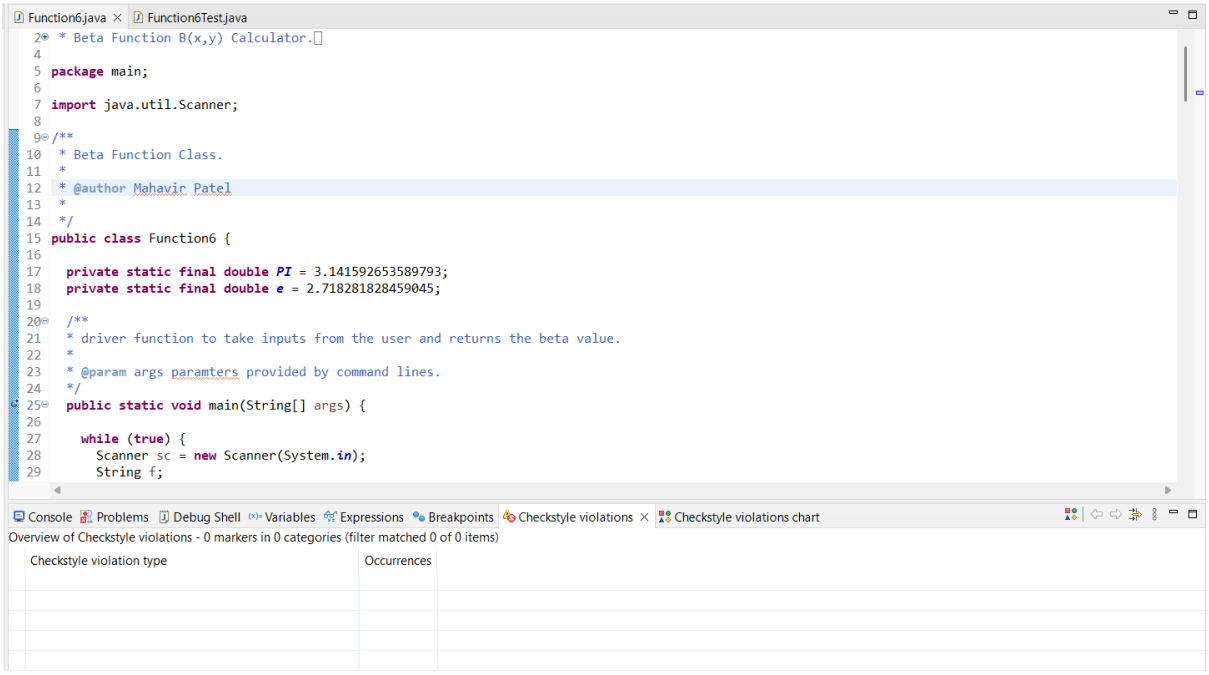

Figure 5.2: Snapshot of the Eclipse CheckStyle plugin

5.3 PMD - Pragmatic Quality Tool

PMD, or programming mistake detector is a source code analyzer. It detects typical programming errors such as unnecessary variables, empty catch blocks, It detects typical programming errors such as unnecessary variables, empty catch blocks, unneccesary object creation, etc. The code error seen in the following screenshot was found in Ecplise IDE's source code and was verified using the PMD plugin.

Advantages

- By examining unnecessary variables or objects, space efficiency is increased and code analysis can be done.
- The copy-paste detector, or CPD, is offered. In the source code, redundant code is discovered.
- Multiple programming languages, markup languages, Salesforce.com Apex and Visualforce, PLSQL, Apache Velocity, XML, XSL, and other technologies are supported.
- Additionally PMD allows user-defined rules for source code analysis

Disadvantages

- There are times when these programmes generate false positives and false negatives, providing the appearance that everything is being addressed.
- Since it is more difficult to identify the precise location of the code vulnerability, a solution takes longer to implement. Vulnerabilities in the runtime environment are not found.

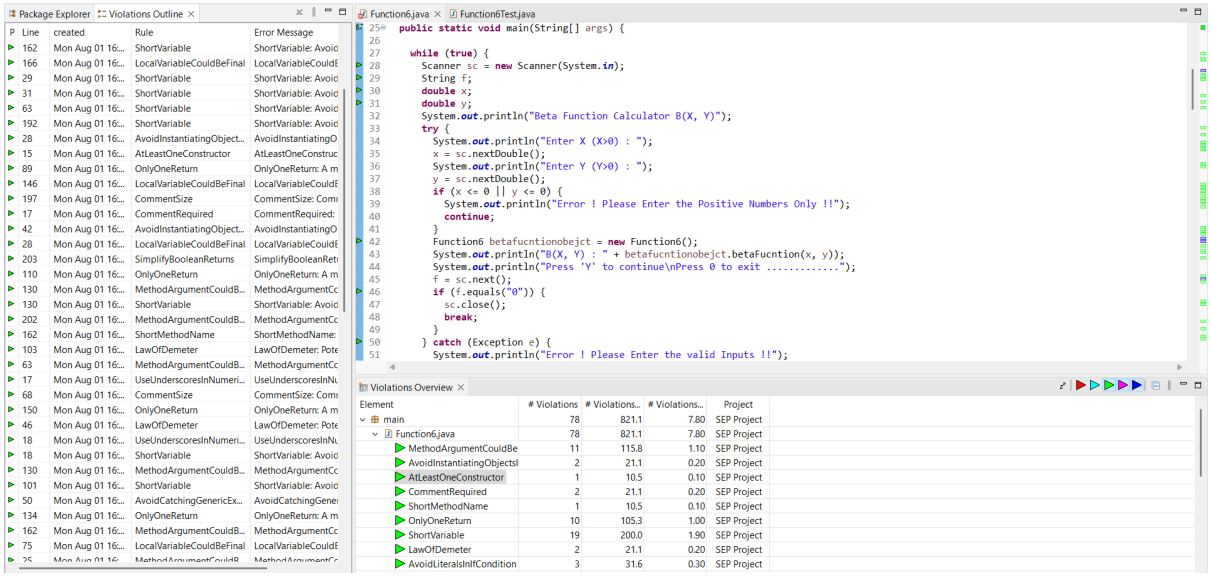

Figure 5.3: Snapshot of the Eclipse PMD plugin

5.4 Quality Attributes

5.4.1 Usability

The beta function calculator has a straightforward user interface that is easy enough for non-technical people to understand. Additionally, it offers an error message that any user may readily understand, such as improper inputs or beta function values such as negative or zero. Even for small values of integers, the Stirlings Approximation offers the user confidence in the accuracy of the solution.

5.4.2 Maintainability

Due to the separation of various tasks, the beta function calculator is simple to maintain without causing any system disruptions or crashes. Additionally, the user interface might be included. Furthermore, it may be adjusted to account for changes in the future and can accommodate various approaches to determine the beta values of the inputs.

5.4.3 Correctness

The results from this calculator are correct for integer numbers. However, it delivers an accurate answer with an accuracy of up to 7 decimal places for fractional integers. Even with a small number, Stirling's approach produces almost exact results, but as the number increases, the method gives a user the right answers.

5.4.4 Efficiency

In both space and time, the calculator application is incredibly effective. It reduces the Stack-Overflow error using the tail recursive method and provides correct results in a short amount of time.

5.4.5 Robustness

The beta calculator is really reliable. It offers an error message that the user may readily understand. If invalid inputs are given, the program will continue to run without pausing the process and will give an error message based on the circumstances that led to the disruption in the flow.

5.5 Error Message and Exception Handling

Responding to undesirable or unexpected events that occur while a computer program is running is known as exception handling. Without this procedure, exceptions would disrupt a program's normal operation and lead to a crash. To avoid this, exception handling deals with these occurrences.

In Beta Function Calculator, if a user enters a number other than positive real numbers, then the user will be notified by an error message "Please enter the positive numbers only"

Moreover, if the user enters a String as input then the system throws an exception containing a message "Please enter the valid inputs" and for any general exception a message "Fatal error" is thrown by an exception handling mechanism.

6 Problem 5

Requirements Traceability

Here given below is the JUnit test case report to check all the requirement has been satisfied.

1. Test Case

- **Test Case ID :** TC1
- **Requirement ID :** FR1, FR2, FR3, FR7
- **Test Case Method :** testMain()
- **Description :** The testMain() function verifies if the user has supplied accurate and legitimate inputs. If not it will provide the Error message and handles the exception. It also check the inputs are not zero or negative or string or character.
- **Test Inputs :** (9,9), (-4,-4), (-4,0), (1,3)
- **Test Expected Outcome :** 4.570592805886924E-6, Error Message, Error Message, 0.3333333333333333
- **Test Actual Outcome :** 4.570592805886924E-6, Error Message, Error Message, 0.3333333333333333
- **Result :** Success

2. Test Case

- **Test Case ID :** TC2
- **Requirement ID :** FR4, FR5, FR7
- **Test Case Method :** testBetaFunction()
- **Description :** The testBetaFunction() checks the inputs and and calculate the beta values with the help of gamma beta function relation.
- **Test Inputs :** (1,1), (100,1)
- **Test Expected Outcome :** 1, 0.009999999999999998
- **Test Actual Outcome :** 1, 0.009999999999999998
- **Result :** Success

3. Test Case

- **Test Case ID :** TC3
- **Requirement ID :** FR4, FR6
- **Test Case Method :** testBetaFunctionFraction()
- **Description :** The testBetaFunctionFraction() calculate the beta values for the fractional numbers using the stirling's approximation because the factorial of the decimal number is not defined.
- **Test Inputs :** (15.8, 20.8)
- **Test Expected Outcome :** 1.1488056591230658E-11
- **Test Actual Outcome :** 1.1488056591230658E-11
- **Result :** Success

4. Test Case

- **Test Case ID :** TC4
- **Requirement ID :** FR7
- **Test Case Method :** testBetaFunctionZero()
- **Description :** The testBetaFunctionFraction() check the error message and validate the exception if one the input value is zero.
- **Test Inputs :** (0,0), (0,5), (5,0)
- **Test Expected Outcome :** Exception and Error Message
- **Test Actual Outcome :** Exception and Error Message
- **Result :** Success

5. Test Case

- **Test Case ID :** TC5
- **Requirement ID :** FR7
- **Test Case Method :** testBetaFunctionNegative()
- **Description :** The testBetaFunctionNegative() check if the negative number is provided then the function does not stop and exception are handled properly.
- **Test Inputs :** (-5,-5), (0,-5), (-5,0)
- **Test Expected Outcome :** Exception and Error Message
- **Test Actual Outcome :** Exception and Error Message
- **Result :** Success

6. Test Case

- **Test Case ID :** TC6
- **Requirement ID :** FR4, FR5
- **Test Case Method :** testWholeNumber()
- **Description :** The test method testWholeNumber() checks the given inputs is whole number or the fraction.
- **Test Inputs :** 0.12, 789.646, 12.00, 45
- **Test Expected Outcome :** False, False, True, True
- **Test Actual Outcome :** False, False, True, True
- **Result :** Success

Bibliography

[1] Beta Function WikiPedia
https://en.wikipedia.org/wiki/Beta_function

[2] Beta Function Properties
https://byjus.com/maths/beta-function/

[3] Stirling's Approximation
https://en.wikipedia.org/wiki/Stirling%27s_approximation

[4] Tail Recursion
https://www.geeksforgeeks.org/tail-recursion/

[5] Error Handling Defination
https://www.techtarget.com/searchsoftwarequality/definition/error-handling

[6] PMD
https://pmd.github.io/latest/index.html

[7] Debugging in the Eclipse IDE
eclipse.org/community/eclipse_newsletter/2017/june/article1.php

[8] Quality attributes in software architecture
https://syndicode.com/blog/12-software-architecture-quality-attributes/

[9] Google's Software Architecture and Quality Attributes
https://sites.google.com/site/misresearch000/home/software-architecture-quality-attributes

[10] Google Checkstyle for JAVA
https://checkstyle.sourceforge.io/google_style.html