SOEN-6011 PROJECT

Eternity

Akhil Sharma

Implementation of the Power Function.

Contents

0.1 Introduction

The objective of this medium-sized scientific software engineering project is to implement the power function while approaching the problem from first principles. Tasks like **documenting**, **versioning**, and **testing** are considered first class citizens.

This report will first discuss about the power function. This is followed by defining the requirements for the system. Defining the requirements will also impose a restriction on the scope of the project. This is followed by comparing different algorithms for calculating the power function and selecting the most efficient based on the time complexity. Next, the method is implemented from scratch in java along with a simple textual user interface. The report also discusses the use of various tools like the debugger and the type check. Finally, the report will discuss the process of testing the system.

As required by the project description, this report was typeset using LaTeX. The LaTeXsource was tested online at Overleaf and offline on MikTex for windows.

The source code for the project is hosted on GitHub

The terms Project, System, Software system, Eternity all refer to the same thing.

Chapter 1

Power Function

Definition 1.0.1. The **power function** is defined as the function that takes any numbers x and y as input, raises x to y, and returns x^y as output.

The power function is a transcendental function and cannot be expressed in terms of a finite sequence of the algebraic operations of raising to a power, division, multiplication, addition, subtraction, and root extraction wikipedia.

All the exponentiation rules are applicable here.

1.1 Domain and Co-domain

The domains and co-domains of x and y vary as follows [1]:

- If **base x is a positive real number**, then y belongs to the set of all reals numbers.

  The corresponding range is the set of all positive real numbers.

- If **base x is zero**, then y belongs to the set of non-negative real numbers.

  This is because a negative y would lead to division by zero.

- If **base x is negative**, then y may only have to certain values:

  - y **may be** any integer.
  - y **may be** a fraction of the form a/b where b is odd.
  - y **may not be** a fraction of the form a/b where b is even.
  - y **may not be** an irrational number.

1.2 Examples

Let's look a few examples:

Any number raised to the power one results in the same number, $2^1 = 2$.

Two to the power three means two times two times two or $2^3 = 8$.

Zero raised to any positive power is 0.

Zero raised to power zero is 1.

For negative powers, $x^{-y} = 1/x^y$. So, $2^{-1} = 1/2^1 = 0.5$

1.3 Context of use Model

The context of use is a description of the conditions under which the software system will be used under the normal working circumstances. A context of use model is a useful tool to explore and understand the details and boundaries of a project. [2]

There is currently no standard for representing the context of use. A context diagram is however a suitable candidate for mind mapping.

Fig 1. is a context diagram for this project built using the GitMind tool. It focuses on the users of the systems and the environments where this system will be operated. The users block is subdivided into blocks representing the potential users of the system and their requirements for effective and efficient completion of their responsibilities.

The environment block explores the different system environments needed to perform different tasks. In particular, the developers, testers and, maintainer require a Technical environment. The end-user however doesn't require anything more than the general environment.

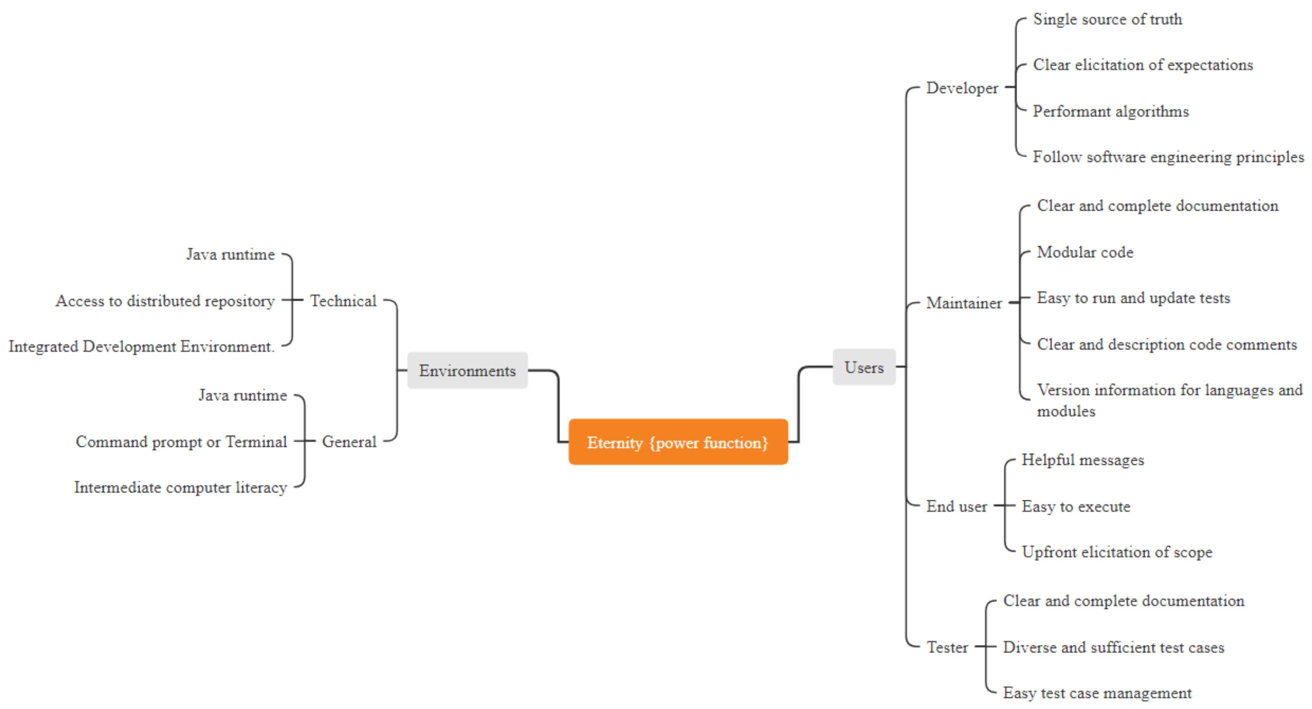

Eternity {power function}
Environments
Technical
Java runtime
Access to distributed repository
Integrated Development Environment.
General
Java runtime
Command prompt or Terminal
Intermediate computer literacy
Users
Developer
Single source of truth
Clear elicitation of expectations
Performant algorithms
Follow software engineering principles
Maintainer
Clear and complete documentation
Modular code
Easy to run and update tests
Clear and description code comments
Version information for languages and modules
End user
Helpful messages
Easy to execute
Upfront elicitation of scope
Tester
Clear and complete documentation
Diverse and sufficient test cases
Easy test case management

Figure 1.1: Context of use for Eternity.

Chapter 2

Requirements

This chapter lists the software requirements using the format outlined in the ISO/IEC/IEEE 29148 sections 5.2.4 to 5.2.7 (“ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering,” 2018). The following meta data is associated with each requirement:

ID This string uniquely identifies the requirement.

VERSION This indicates the number of times a requirement has been updated.

PRIORITY This string indicates the importance of implementing this requirement for the minimum viable product. Priority may be one of HIGH, AVERAGE, or LOW.

TYPE This string indicates whether the requirement is functional or non-functional in nature.

The requirements were managed using the Jira dashboard. See figure 4.1.

ID	VERSION	PRIORITY	TYPE
R1	1	HIGH	NON-FUNCTIONAL
Eternity shall execute on any device with JVM.			

Table 2.1: Requirement 1

ID	VERSION	PRIORITY	TYPE
R2	1	HIGH	FUNCTIONAL
Eternity shall allow multiple calculations per session			

Table 2.2: Requirement 2

ID	VERSION	PRIORITY	TYPE
R3	1	HIGH	NON-FUNCTIONAL
Eternity shall be developed with maintainability in mind.			

Table 2.3: Requirement 3

ID	VERSION	PRIORITY	TYPE
R4	1	HIGH	NON-FUNCTIONAL
Eternity shall be modular and testable.			

Table 2.4: Requirement 4

ID	VERSION	PRIORITY	TYPE
R5	1	HIGH	NON-FUNCTIONAL
Eternity shall have a simple user interface to increase usability.			

Table 2.5: Requirement 5

ID	VERSION	PRIORITY	TYPE
R6	1	HIGH	FUNCTIONAL
Eternity shall allow only integer values of y when calculating x^y.			

Table 2.6: Requirement 6

ID	**VERSION**	**PRIORITY**	**TYPE**
R7	1	HIGH	FUNCTIONAL
Eternity shall allow all real values for x when calculating x^y.			

Table 2.7: Requirement 7

ID	**VERSION**	**PRIORITY**	**TYPE**
R8	1	HIGH	NON-FUNCTIONAL
Eternity shall provide a help menu for users.			

Table 2.8: Requirement 8

ID	**VERSION**	**PRIORITY**	**TYPE**
R9	1	HIGH	NON-FUNCTIONAL
Eternity shall provide Javadoc documentation.			

Table 2.9: Requirement 9

ID	**VERSION**	**PRIORITY**	**TYPE**
R10	1	HIGH	FUNCTIONAL
Eternity shall use optimal algorithms.			

Table 2.10: Requirement 10

ID	**VERSION**	**PRIORITY**	**TYPE**
R11	1	HIGH	FUNCTIONAL
Eternity shall not allow x to be 0 when y is negative.			

Table 2.11: Requirement 11

ID	**VERSION**	**PRIORITY**	**TYPE**
R12	1	HIGH	FUNCTIONAL
Eternity shall return 1 when the power is 0.			

Table 2.12: Requirement 12

ID	**VERSION**	**PRIORITY**	**TYPE**
R13	1	HIGH	FUNCTIONAL
Eternity shall return x when $y = 1$ in x^y.			

Table 2.13: Requirement 13

ID	**VERSION**	**PRIORITY**	**TYPE**
R14	1	LOW	FUNCTIONAL
Eternity will maintain a history of previous operations.			

Table 2.14: Requirement 14

Chapter 3

Algorithm

There exist a few algorithms for calculating the power function. However, requirements 2.6 and 2.7 significantly reduce the scope this project. We will consider two algorithms for Eternity, the repeated multiplication or iterative solution and, the divide and conquer solution. When discussing these algorithms we will assume that the power y is positive since the solution for negative y can be derived trivially from this value.

3.1 Iterative Solution (Naive approach)

The iterative solution is the simplest way solve this problem. The calculation of x^y involves multiplying X with itself exactly y times. For example, $2^3 = 2 * 2 * 2 = 8$.

3.1.1 Pseudocode

Input : Numbers x and y where x is a real number and y is a positive integer.

Output : Return the value power function x^y.

```
Function: $power\_incremental(x, y)$
$power \leftarrow 1$;
while $y > 0$ do
    $power \leftarrow power * x$;
    $y = y - 1$;
end
return $power$;
```

Algorithm 1: Power function using repeated multiplication (Naive).

3.1.2 Complexity

This is a relatively straightforward algorithm and runs in linear time. This results to a time complexity of $O(n)$ where n is the magnitude of power y.

3.2 Divide and Conquer

In the **Divide and Conquer** approach the problem is defined recursively as follows:

$$power(x, y) = power(x, y/2) * power(x, y/2), \textit{ if } y \textit{ is even}$$

$$power(x, y) = x * power(x, y/2) * power(x, y/2), \textit{ if } y \textit{ is odd}$$

Here, the problem is being divided into a sub-problem which is half the size of the original problem until the base case is reached where $y = 1$.

3.2.1 Pseudocode

Input : Numbers x and y where x is a real number and y is a positive integer.

Output : Return the value power function x^y.

```
Function: $power\_divide\_and\_conquer(x, y)$
if $y = 0$ then
    return 1;
end
$pow \leftarrow power\_divide\_and\_conquer(x, y/2)$;
if $y$ is odd then
    return $x * pow * pow$;
end
return $pow * pow$;
```

Algorithm 2: Power function using divide and conquer.

3.2.2 Complexity

Since this algorithm continuously divides the problem in half we get the time complexity of $\mathcal{O}(\log n)$ where n is the magnitude of y.

3.3 Final Selection

By comparing the above algorithms, it is clear that the algorithm 2 **divide and conquer** performs better than algorithm 1 **direct approach**. As a result, Eternity will implement the second algorithm or algorithm 2.

3.4 Pseudocode Style Selection

There exist a number of styles for writing pseudocodes. Mind mapping was used to determine the style that suits our purpose. See figure 3.1.

The pseudocode style provided by the algorithm2e LaTeXpackage was determined to be best suited since it provides a format which is very similar to the c-style or java-style and capable of representing much detail.

This style was preferred over the python style since it is **closer in distance** to the java code in the repository.

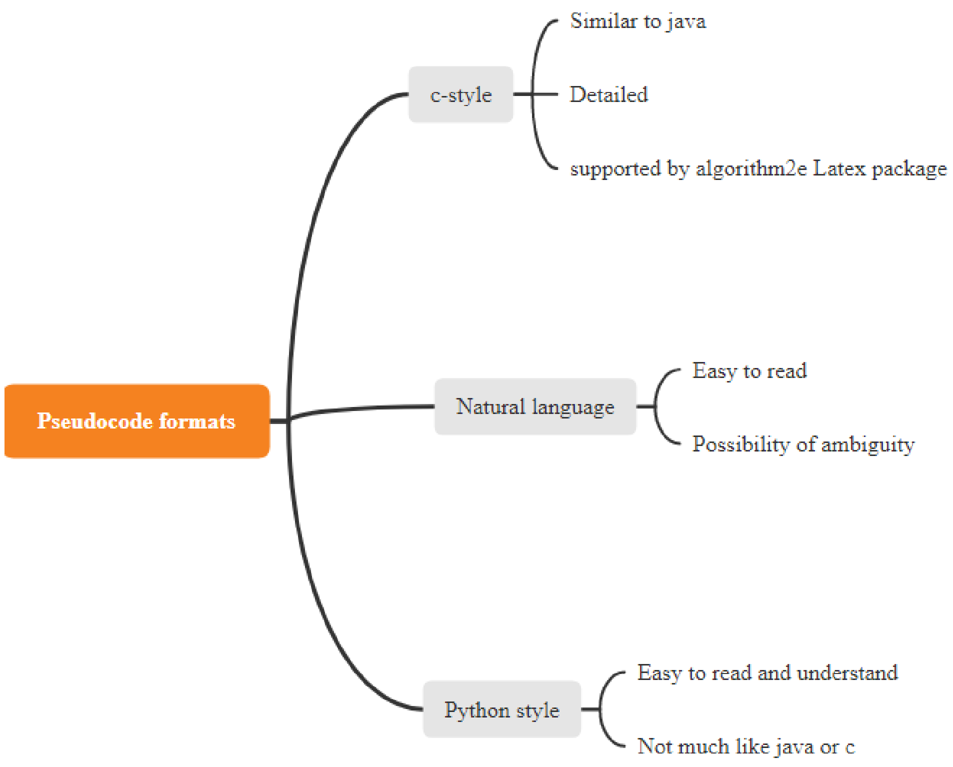


Figure 3.1: Mind Mapping to determine the pseudocode style.

Chapter 4

Problem

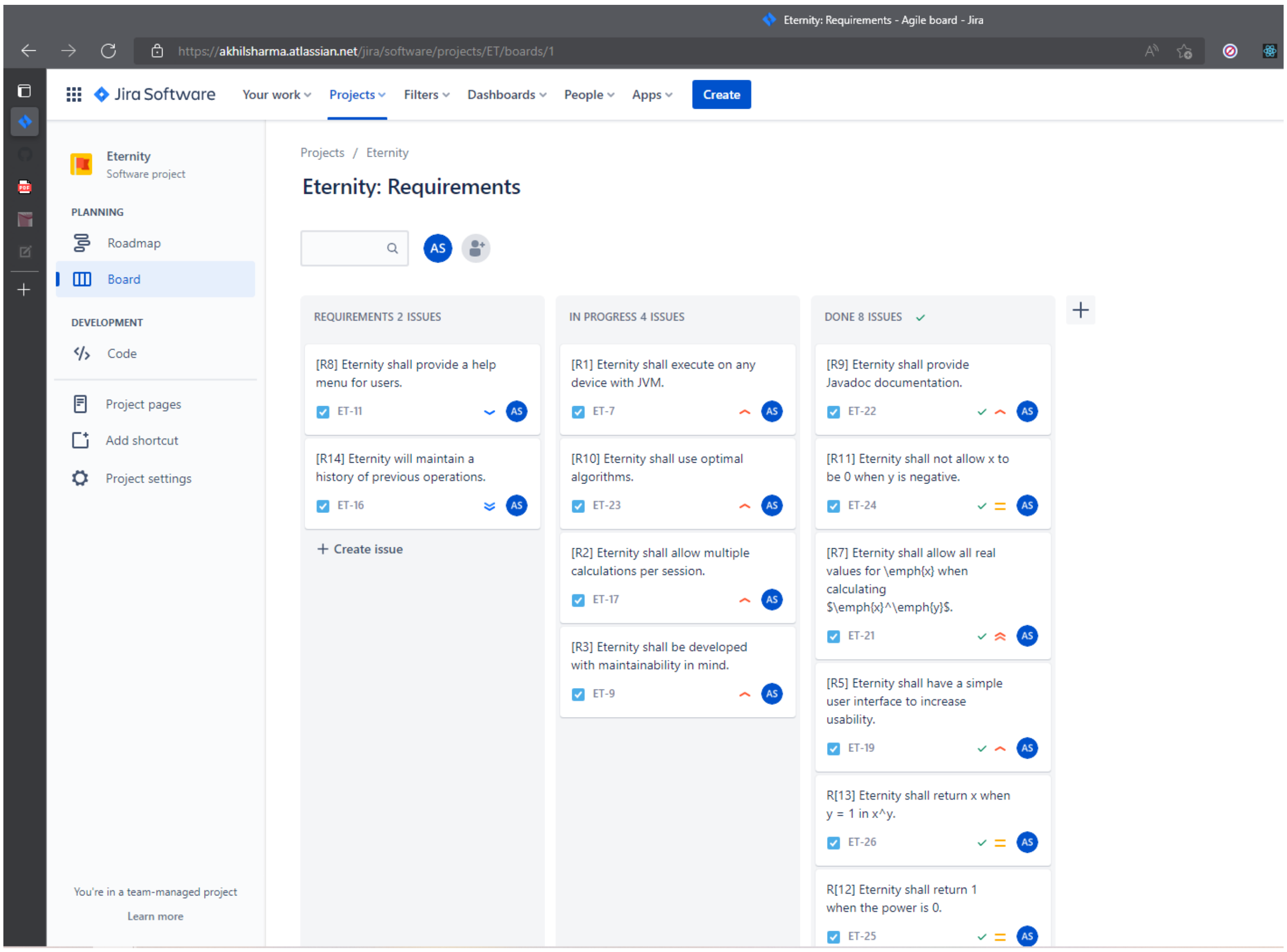
The project is implemented from scratch in Java. The term “scratch” means that, apart from the functions related to input, output, arithmetic, and user interface design, your implementation does not use any built-in or library functions provided by Java.

The project uses Google’s java style guide since most of the popular style guides are derived from it.

Eternity uses a textual interface since seemed appropriate for a project of this scope. The developer’s inexperience in building graphical user interface also contributed to this decision.

4.1 Requirements Management

During the development of Eternity the requirements were managed using Jira. Figure 4.1 the Jira dashboard for requirements.



Eternity: Requirements

REQUIREMENTS 2 ISSUES

[R8] Eternity shall provide a help menu for users. ET-11

[R14] Eternity will maintain a history of previous operations. ET-16

IN PROGRESS 4 ISSUES

[R1] Eternity shall execute on any device with JVM. ET-7

[R10] Eternity shall use optimal algorithms. ET-23

[R2] Eternity shall allow multiple calculations per session. ET-17

[R3] Eternity shall be developed with maintainability in mind. ET-9

DONE 8 ISSUES

[R9] Eternity shall provide Javadoc documentation. ET-22

[R11] Eternity shall not allow x to be 0 when y is negative. ET-24

[R7] Eternity shall allow all real values for \emph{x} when calculating $\emph{x}^\emph{y}$. ET-21

[R5] Eternity shall have a simple user interface to increase usability. ET-19

R[13] Eternity shall return x when y = 1 in x^y. ET-26

R[12] Eternity shall return 1 when the power is 0. ET-25

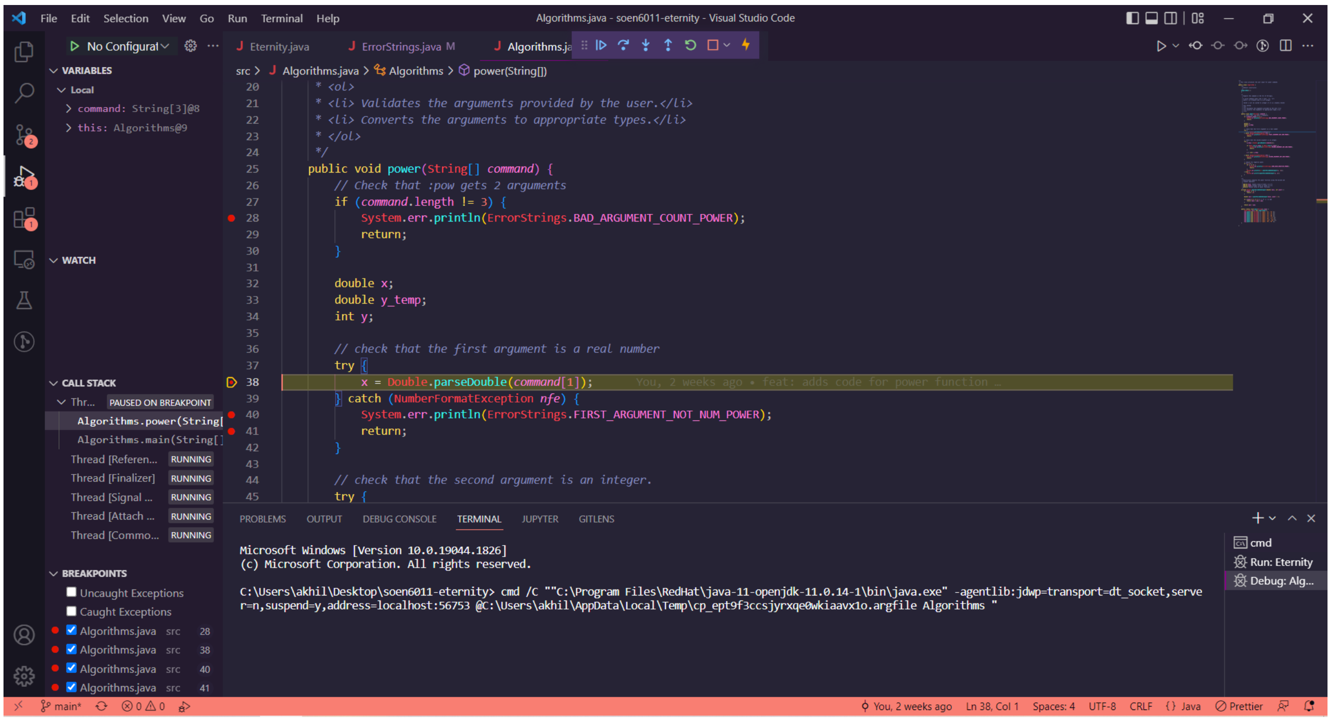
Figure 4.1: Requirements management in jira.

4.2 Debugger

Eternity uses the open-source Debugger for Java [] built into visual studio code. It is a lightweight java debugger based on Java Debug Server which extends the Language Support for Java by Red Hat. It allows users to debug Java code using Visual Studio Code (VS Code). It has a number of features like:

- Launch/Attach
- Breakpoints/Conditional Breakpoints/Logpoints
- Exceptions
- Pause and Continue
- Step In/Out/Over
- Variables
- Callstacks
- Threads
- Debug console
- Evaluation
- Hot Code Replace

Figure 4.2 shows the debugger in action. It is an open-source project backed by tech giants like Microsoft. It offer a very granular control over the process and can be configured as needed. It comes pre-installed with VS Code and needs little knowledge to get started with. All these factors made it the debugger of choice for me.



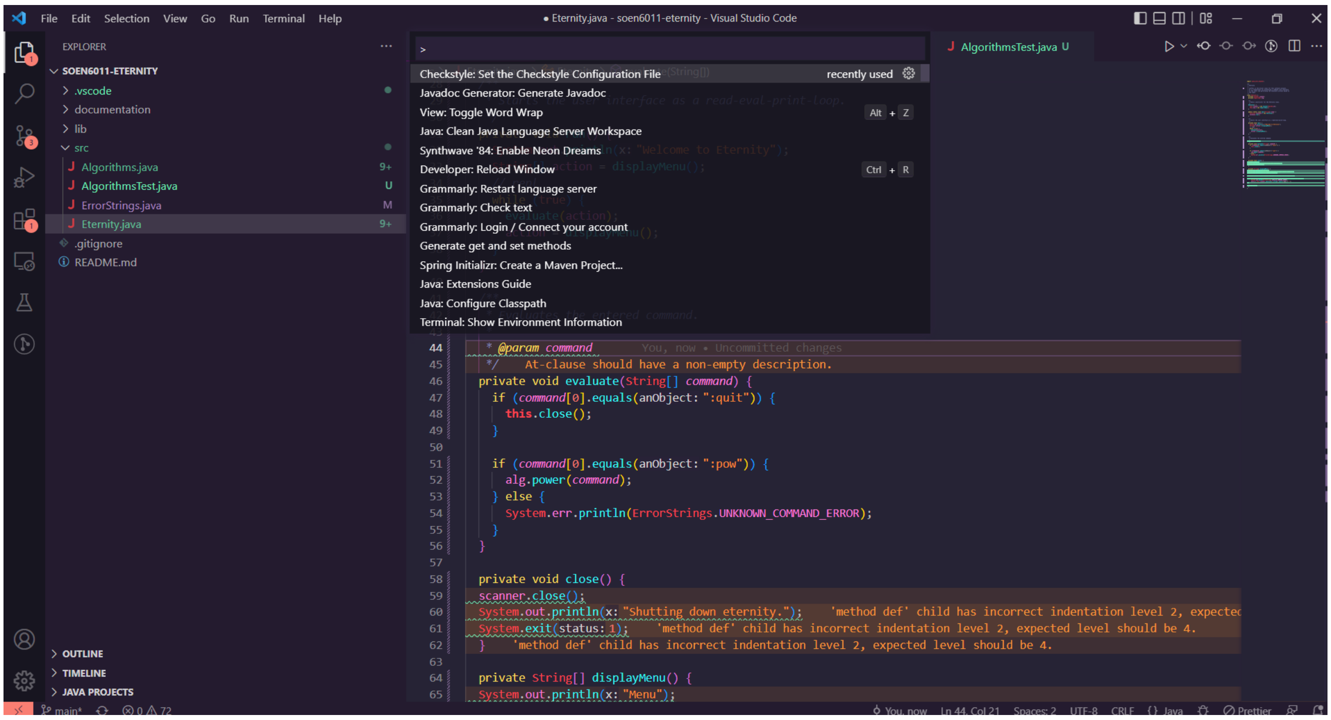
```java
    * <ol>
    * <li> Validates the arguments provided by the user.</li>
    * <li> Converts the arguments to appropriate types.</li>
    * </ol>
    */
    public void power(String[] command) {
        // Check that :pow gets 2 arguments
        if (command.length != 3) {
            System.err.println(ErrorStrings.BAD_ARGUMENT_COUNT_POWER);
            return;
        }

        double x;
        double y_temp;
        int y;

        // check that the first argument is a real number
        try {
            x = Double.parseDouble(command[1]);     You, 2 weeks ago • feat: adds code for power function …
        } catch (NumberFormatException nfe) {
            System.err.println(ErrorStrings.FIRST_ARGUMENT_NOT_NUM_POWER);
            return;
        }

        // check that the second argument is an integer.
        try {
```

```
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\akhil\Desktop\soen6011-eternity> cmd /C ""C:\Program Files\RedHat\java-11-openjdk-11.0.14-1\bin\java.exe" -agentlib:jdwp=transport=dt_socket,serve
r=n,suspend=y,address=localhost:56753 @C:\Users\akhil\AppData\Local\Temp\cp_ept9f3ccsjyrxqe0wkiaavx1o.argfile Algorithms "
```

Figure 4.2: VS Code debugger in action.

4.3 Checkstyle

Checkstyle is a convenient tool to apply Checkstyle rules to your Java source code so you can see the style issues and fix them on the fly. It automates the process of checking your Java code so the developer is freed while keeping your format correct. The checkstyle plugin for vs code also includes the Google style guide built-in to it make the whole process very smooth. Figure 4.3 show the checkstyle tool in action.



```
>
Checkstyle: Set the Checkstyle Configuration File
Javadoc Generator: Generate Javadoc
View: Toggle Word Wrap
Java: Clean Java Language Server Workspace
Synthwave '84: Enable Neon Dreams
Developer: Reload Window
Grammarly: Restart language server
Grammarly: Check text
Grammarly: Login / Connect your account
Generate get and set methods
Spring Initializr: Create a Maven Project...
Java: Extensions Guide
Java: Configure Classpath
Terminal: Show Environment Information
```

```
   * @param command        You, now • Uncommitted changes
   */    At-clause should have a non-empty description.
  private void evaluate(String[] command) {
    if (command[0].equals(anObject: ":quit")) {
      this.close();
    }

    if (command[0].equals(anObject: ":pow")) {
      alg.power(command);
    } else {
      System.err.println(ErrorStrings.UNKNOWN_COMMAND_ERROR);
    }
  }

  private void close() {
  scanner.close();
  System.out.println(x: "Shutting down eternity.");    'method def' child has incorrect indentation level 2, expected
  System.exit(status: 1);    'method def' child has incorrect indentation level 2, expected level should be 4.
  }    'method def' child has incorrect indentation level 2, expected level should be 4.

  private String[] displayMenu() {
  System.out.println(x: "Menu");
```

Figure 4.3: Running checkstyle on code using the built-in Google style guide.

4.4 Quality Attributes Management

A significant consideration went into managing the quality attributes of Eternity. Some of these attributes and associated actions mentioned below.

- **Space efficiency**: Eternity evaluates all commands on the fly without saving artifact from the previous executions. Variable creation minimized when possible. Efforts have been made to ensure graceful exiting by closing streams before quitting.

- **Portability**: Eternity uses an unmanaged directory structure with out any build tool written in java from scratch. This results in high portability as the code will compile and execute on any device which which has JVM installed. Special care has been taken to prevent dependence on any IDE or operating system as well.

- **Maintainability**: Many actions have been taken to make the project maintainable. Extensive use of javadoc comments, breaking down functionality into separate files, maintaining a commit history using distributed version management are some of these actions. In addition, each command is validated and handled by a separate file making it trivial to add new commands.

- **Correctness**: The correctness has be ensured by adding a sufficient number of test cases for unit testing the commands (here, power).

- **Time efficiency**: The algorithm used by Eternity is based on the 'divide and conquer approach' and was selected due to its superior time complexity compare to the naive algorithms. This contributes to improving the time efficiency of the project.

- **Robustness**: Extensive unit testing, **exception handling** help make eternity robust.

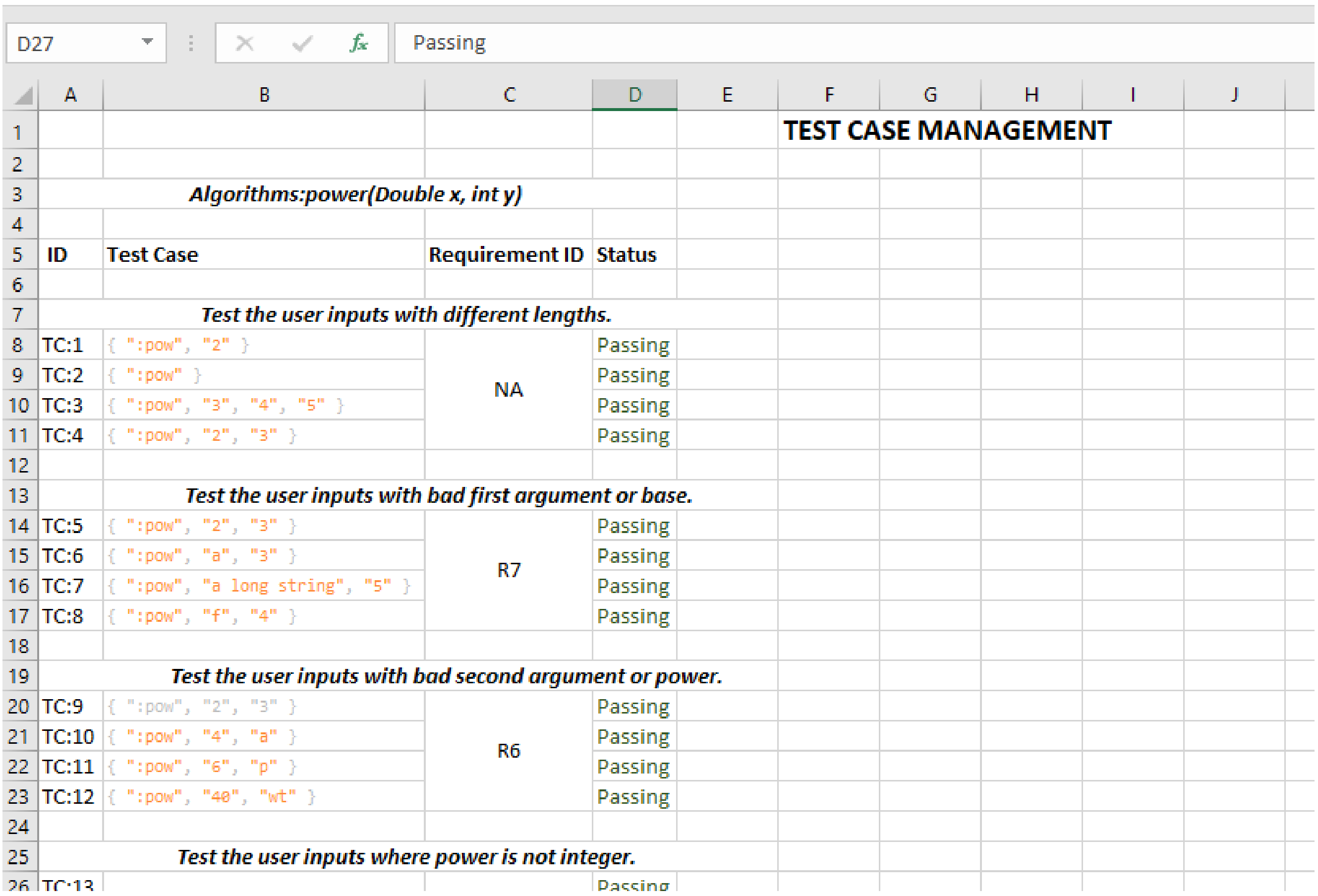
- **Usability**: Eternity offers a very simple textual interface which display the available commands to the user. In addition it also displays **helpful messages (errors or otherwise)** to make sure that the user is never in the dark. User is also greeted with a welcome and an exit message to provide the boundary to the user experience. All these efforts contribute to the usability of the product.

Chapter 5

Testing

Testing is necessary to ensure the correctness and robustness of the software system. Unit testing for Eternity was done using the JUnit 4 library. Expiration of Jira trial period motivated the use of Microsoft Excel for the purpose of Test case management. Figure 5.1 shows a snapshot of the test case management workbook.

The association between Test cases and the requirements were also recorded.



D27 | Passing

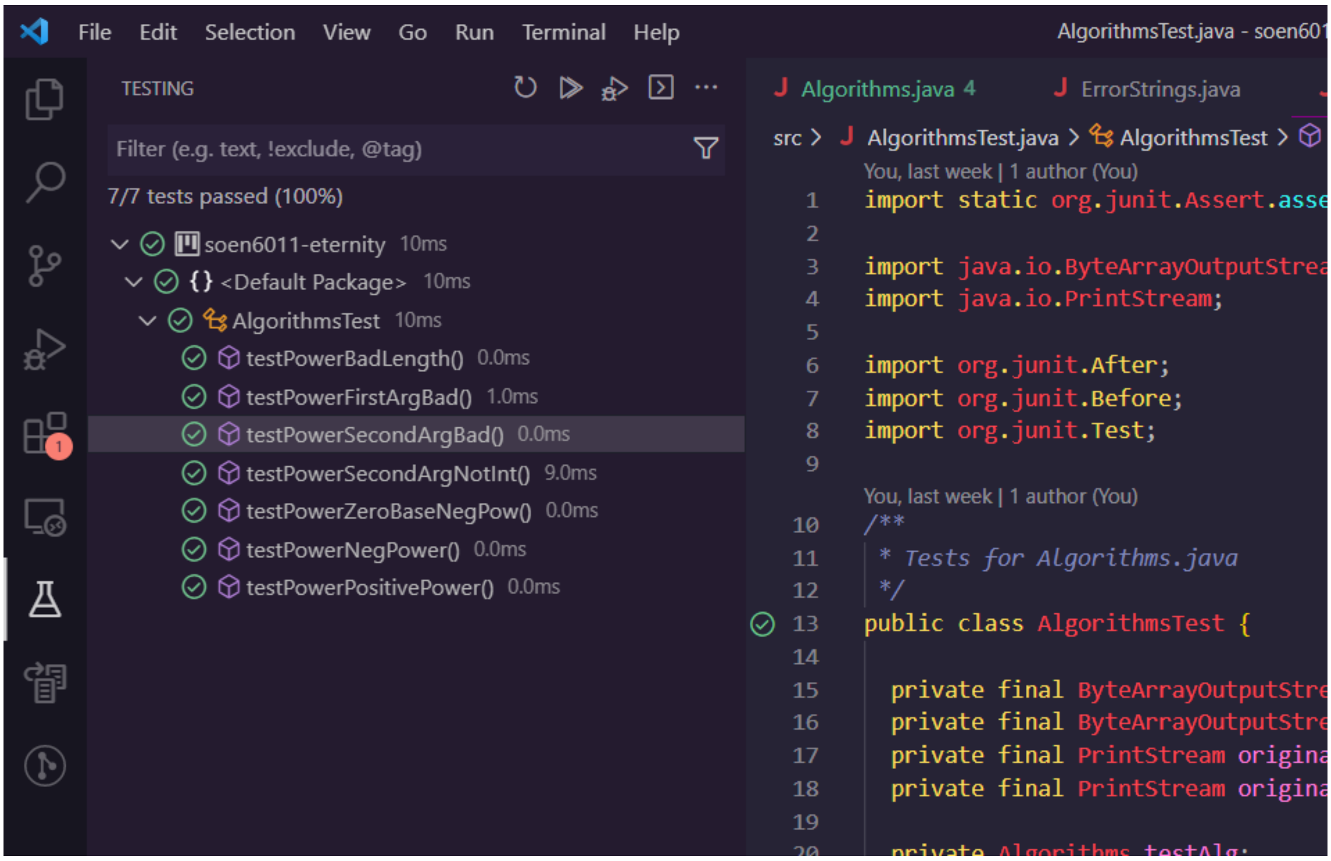
TEST CASE MANAGEMENT

Algorithms:power(Double x, int y)

ID	Test Case	Requirement ID	Status
	Test the user inputs with different lengths.		
TC:1	`{ ":pow", "2" }`	NA	Passing
TC:2	`{ ":pow" }`		Passing
TC:3	`{ ":pow", "3", "4", "5" }`		Passing
TC:4	`{ ":pow", "2", "3" }`		Passing
	Test the user inputs with bad first argument or base.		
TC:5	`{ ":pow", "2", "3" }`	R7	Passing
TC:6	`{ ":pow", "a", "3" }`		Passing
TC:7	`{ ":pow", "a long string", "5" }`		Passing
TC:8	`{ ":pow", "f", "4" }`		Passing
	Test the user inputs with bad second argument or power.		
TC:9	`{ ":pow", "2", "3" }`	R6	Passing
TC:10	`{ ":pow", "4", "a" }`		Passing
TC:11	`{ ":pow", "6", "p" }`		Passing
TC:12	`{ ":pow", "40", "wt" }`		Passing
	Test the user inputs where power is not integer.		
TC:13			Passing

Figure 5.1: Test case management using excel.

Figure 5.2 demonstrates the execution of the JUnit-4 test cases in VS Code.
All the tests are available in this file on github.


File
Edit
Selection
View
Go
Run
Terminal
Help
AlgorithmsTest.java - soen601
TESTING
Filter (e.g. text, !exclude, @tag)
Algorithms.java 4
ErrorStrings.java
src > AlgorithmsTest.java > AlgorithmsTest >

7/7 tests passed (100%)

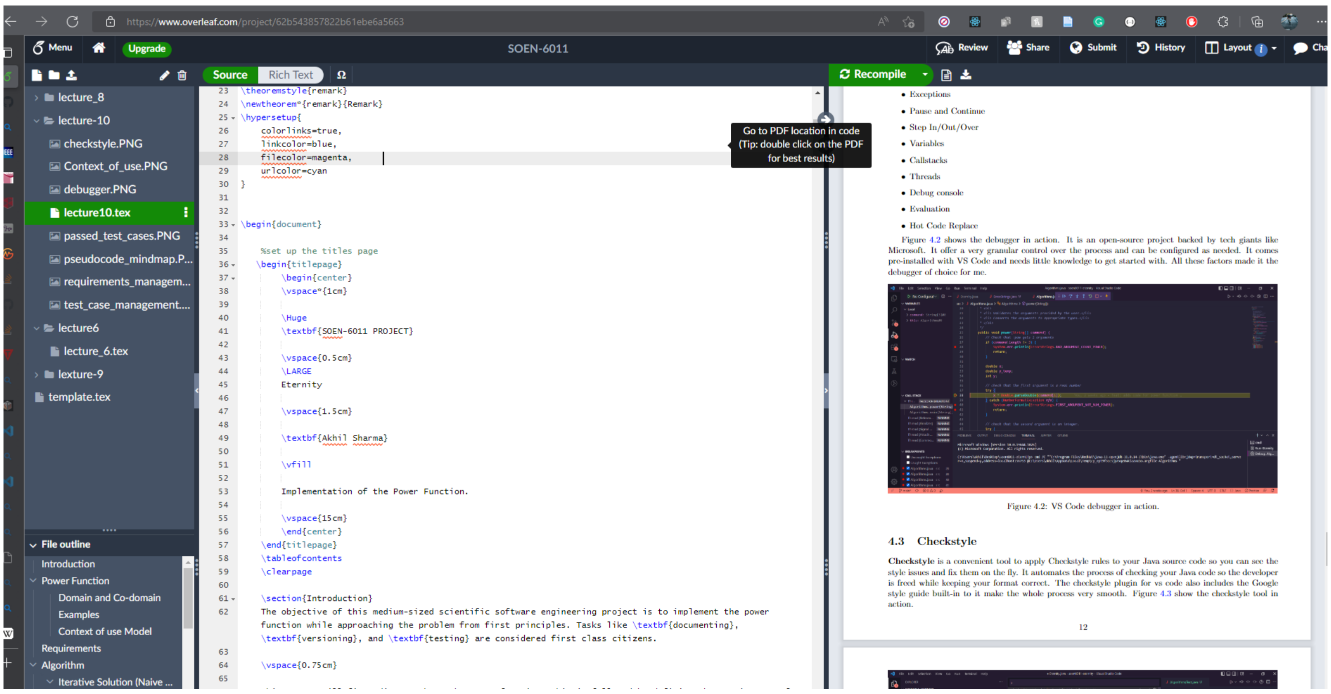
```
soen6011-eternity  10ms
  {} <Default Package>  10ms
    AlgorithmsTest  10ms
      testPowerBadLength()  0.0ms
      testPowerFirstArgBad()  1.0ms
      testPowerSecondArgBad()  0.0ms
      testPowerSecondArgNotInt()  9.0ms
      testPowerZeroBaseNegPow()  0.0ms
      testPowerNegPower()  0.0ms
      testPowerPositivePower()  0.0ms
```

```
You, last week | 1 author (You)
import static org.junit.Assert.asse

import java.io.ByteArrayOutputStrea
import java.io.PrintStream;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

You, last week | 1 author (You)
/**
 * Tests for Algorithms.java
 */
public class AlgorithmsTest {

  private final ByteArrayOutputStre
  private final ByteArrayOutputStre
  private final PrintStream origina
  private final PrintStream origina

  private Algorithms testAlg;
```

Figure 5.2: Execution of test cases on vs code.

Appendix A

Proof of compilation



```latex
\theoremstyle{remark}
\newtheorem*{remark}{Remark}
\hypersetup{
    colorlinks=true,
    linkcolor=blue,
    filecolor=magenta,
    urlcolor=cyan
}


\begin{document}

    %set up the titles page
   \begin{titlepage}
        \begin{center}
        \vspace*{1cm}

        \Huge
        \textbf{SOEN-6011 PROJECT}

        \vspace{0.5cm}
        \LARGE
        Eternity

        \vspace{1.5cm}

        \textbf{Akhil Sharma}

        \vfill

        Implementation of the Power Function.

        \vspace{15cm}
        \end{center}
    \end{titlepage}
    \tableofcontents
    \clearpage

    \section{Introduction}
    The objective of this medium-sized scientific software engineering project is to implement the power
    function while approaching the problem from first principles. Tasks like \textbf{documenting},
    \textbf{versioning}, and \textbf{testing} are considered first class citizens.

    \vspace{0.75cm}
```

- Exceptions
- Pause and Continue
- Step In/Out/Over
- Variables
- Callstacks
- Threads
- Debug console
- Evaluation
- Hot Code Replace

Figure 4.2 shows the debugger in action. It is an open-source project backed by tech giants like Microsoft. It offer a very granular control over the process and can be configured as needed. It comes pre-installed with VS Code and needs little knowledge to get started with. All these factors made it the debugger of choice for me.

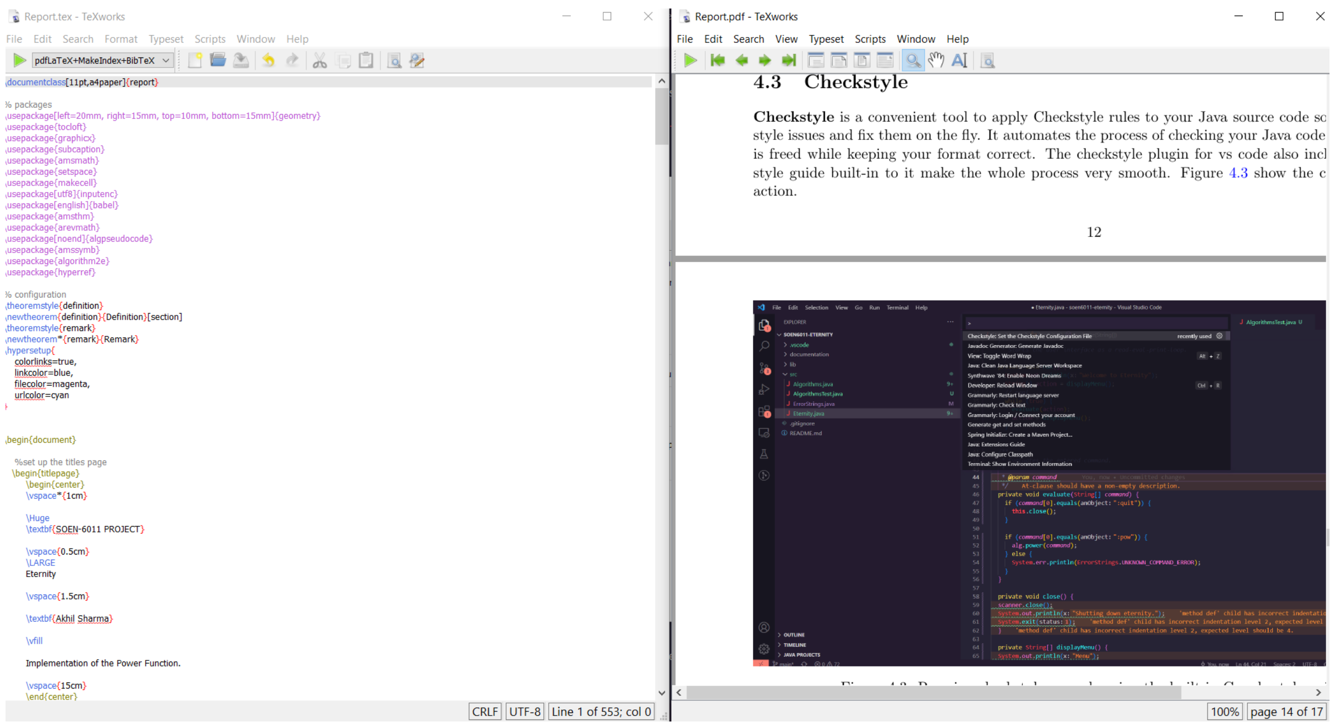
Figure 4.2: VS Code debugger in action.

4.3 Checkstyle

Checkstyle is a convenient tool to apply Checkstyle rules to your Java source code so you can see the style issues and fix them on the fly. It automates the process of checking your Java code so the developer is freed while keeping your format correct. The checkstyle plugin for vs code also includes the Google style guide built-in to it make the whole process very smooth. Figure 4.3 show the checkstyle tool in action.

12

Figure A.1: Proof of compilation online at overleaf.com


Report.tex - TeXworks
File Edit Search Format Typeset Scripts Window Help
pdfLaTeX+MakeIndex+BibTeX

```
\documentclass[11pt,a4paper]{report}

% packages
\usepackage[left=20mm, right=15mm, top=10mm, bottom=15mm]{geometry}
\usepackage{tocloft}
\usepackage{graphicx}
\usepackage{subcaption}
\usepackage{amsmath}
\usepackage{setspace}
\usepackage{makecell}
\usepackage[utf8]{inputenc}
\usepackage[english]{babel}
\usepackage{amsthm}
\usepackage{arevmath}
\usepackage[noend]{algpseudocode}
\usepackage{amssymb}
\usepackage{algorithm2e}
\usepackage{hyperref}

% configuration
\theoremstyle{definition}
\newtheorem{definition}{Definition}[section]
\theoremstyle{remark}
\newtheorem*{remark}{Remark}
\hypersetup{
    colorlinks=true,
    linkcolor=blue,
    filecolor=magenta,
    urlcolor=cyan
}


\begin{document}

    %set up the titles page
  \begin{titlepage}
      \begin{center}
      \vspace*{1cm}

      \Huge
      \textbf{SOEN-6011 PROJECT}

      \vspace{0.5cm}
      \LARGE
      Eternity

      \vspace{1.5cm}

      \textbf{Akhil Sharma}

      \vfill

      Implementation of the Power Function.

      \vspace{15cm}
      \end{center}
```

4.3 Checkstyle

Checkstyle is a convenient tool to apply Checkstyle rules to your Java source code sc
style issues and fix them on the fly. It automates the process of checking your Java code
is freed while keeping your format correct. The checkstyle plugin for vs code also incl
style guide built-in to it make the whole process very smooth. Figure 4.3 show the c
action.

12

Eternity.java - soen6011-eternity - Visual Studio Code
EXPLORER
SOEN6011-ETERNITY
.vscode
documentation
lib
src
Algorithms.java
AlgorithmsTest.java
ErrorStrings.java
Eternity.java
.gitignore
README.md
OUTLINE
TIMELINE
JAVA PROJECTS
Checkstyle: Set the Checkstyle Configuration File
Javadoc: Generator: Generate Javadoc
View: Toggle Word Wrap
Java: Clean Java Language Server Workspace
Synthwave '84: Enable Neon Dreams
Developer: Reload Window
Grammarly: Restart language server
Grammarly: Check text
Grammarly: Login / Connect your account
Generate get and set methods
Spring Initializr: Create a Maven Project...
Java: Extensions Guide
Java: Configure Classpath
Terminal: Show Environment Information

```
   * @param command        You, now • Uncommitted changes
   */   At-clause should have a non-empty description.
  private void evaluate(String[] command) {
    if (command[0].equals(anObject: ":quit")) {
      this.close();
    }

    if (command[0].equals(anObject: ":pow")) {
      alg.power(command);
    } else {
      System.err.println(ErrorStrings.UNKNOWN_COMMAND_ERROR);
    }
  }

  private void close() {
  scanner.close();
  System.out.println(x: "Shutting down eternity.");    'method def' child has incorrect indentatio
  System.exit(status: 1);    'method def' child has incorrect indentation level 2, expected level
  }    'method def' child has incorrect indentation level 2, expected level should be 4.

  private String[] displayMenu() {
  System.out.println(x: "Menu");
```

Figure A.2: Proof of compilation offline on MikTex for windows.

Bibliography

[1] The Algebra Help e-book https://mathonweb.com/help_ebook/html/functions_5.htm#power

[2] What is a Context Diagram – Explain with Examples https://www.edrawmax.com/context-diagram/