**ACKNOWLEDGEMENT**

I am happy to present this report on the internship program. This Program would not have been possible without the guidance, assistance, and suggestions of many individuals. I would like to express my deep sense of gratitude and indebtedness to each and everyone who has helped during my internship program.

I express my heartfelt gratitude and sincere thanks to **Dr.Mohan Babu G . N, Principal of BMSIT&M** for his constant encouragement and inspiration.

I sincerely convey my thanks to **Dr. Thippeswamy, Head of the Department, Computer Science and Engineering, BMSIT&M** for her constant encouragement and support.

I gracefully thank my Internal Guide, **Mrs. Durga Bhavani A, Assistant Professor, Department of Computer Science and Engineering, BMSIT&M** for his/her guidance, support, and advice.

I heartily thank my External Guide, **Mrs. Venkatalaksmi G, Manager,** for his/her constant support, encouragement, and advice.

Finally, I would like to thank my parents, friends, and all those who are involved in the successful completion of the internship program.

**Khushwinder Singh**

**1BY18CS074**

**Internship Letter**

**ABSTRACT**

**Chapter 1**

**About the Company**

**1.1 Vision, Mission and Objectives of the Company**

**Vision:**

To be a world-class enterprise in professional electronics.

**Mission:**

To be a customer focussed, globally competitive company in defence electronics and in other chosen areas of professional electronics, through quality, technology and innovation.

**Objectives:**

* To be a customer focussed company providing state-of-the-art products & solutions at competitive prices, meeting the demands of quality, delivery & service.
* To generate internal resources for profitable growth.
* To attain technological leadership in defence electronics through in-house R&D, partnership with defence/research laboratories & academic institutions.
* To give thrust to exports.
* To create a facilitating environment for people to realise their full potential through continuous learning & teamwork.
* To give value for money to customers & create wealth for shareholders.
* To constantly benchmark the company's performance with best-in-class internationally.
* To raise marketing abilities to global standards.
* To strive for self-reliance through indigenisation.

**1.2 Overview**

Bharat Electronics Limited is an Indian state-owned aerospace and defence company with

about nine factories and several regional offices in India. It is owned by the Indian

Government and primarily manufactures advanced electronic products for the Indian

Armed Forces.

Guided by a farsighted vision to make the country self-reliant in Defence electronics,

Bharat Electronics Limited (BEL) was set up in Bangalore in 1954 by the Government of

India under the Ministry of Defence (MoD). From a humble beginning, BEL has come a

long way. It is now a Navratna PSU and India’s foremost Defence electronics company.

BEL is a multi-product, multi-technology, multi-Unit conglomerate boasting of over 350

products in the areas of Radars, Missile Systems, Military Communications, Naval

Systems, Electronic Warfare & Avionics, C4I Systems, Electro-Optics, Tank Electronics

& Gun/Weapon System Upgrades, Solar Photovoltaic Systems, Electronic Components

and civilian products. The gamut of products includes small components costing a few

rupees to huge systems costing crores of rupees. With its expertise developed over the

years, the company also provides turnkey systems solutions. While Defence continues to

contribute to nearly 80 to 85% of its revenue, BEL has touched a chord with the common

man through civilian products like solar traffic signals and Electronic Voting Machines

(EVMs). BEL’s reliable and tamper-proof EVMs have redefined voting in India,

facilitating free and fair elections. BEL’s customers include the Army, Navy, Air Force,

Paramilitary, Coast Guard, Police, Doordarshan, All India Radio, Department of

Telecommunications and consumers of professional electronic components.

Research & Development has been one of BEL’s core strengths, helping it maintain its

pre-eminence in Defence electronics. The Company has been spending about 8% of its

turnover on R&D. BEL has an extensive infrastructure — its manufacturing network is

spread over 9 Units located across the country. BEL provides after-sales support whenever

and wherever required, irrespective of time or terrain. For instance, its support to the

Defence forces during the Kargil operation was well appreciated.



**Chapter 2**

**About the Department**

**2.1 Software Group**

BEL has an established software team that has for more than 2 decades successfully realised complex, high-quality software solutions. BEL’s software-intensive systems are deployed in challenging operational environments for varied customers including the Indian Defence Forces, Para-Military Forces, Election Commission of India, DRDOs and other PSUs.

The software solutions from BEL display high levels of customisation, integration with legacy and existing systems, and usage of state-of-the-art technology and industry-standard process compliance.

**2.2 Team Objective**

BEL’s Software facility, which was set up in Bengaluru in 1996, primarily for application software development and maintenance of BEL’s own software-intensive projects, has presently grown into a self-sufficient Strategic Business Unit geared to leverage its expertise in both Software Development and Support as well as and Software Assurance Services for domestic and export markets.

BEL presently has the collective software experience of 5000+ person-years and 30+ Million Lines of Code. This includes Software Development and Support projects for the Defence, Para-Military and Civilian sectors. BEL’s Flagship Software Intensive Civilian Products include the Electronic Voting Machine and the Doppler Weather Radar.

The business focus is presently diversified to include software projects for the Defence Market, Non-Defence solutions including the AI (Artificial Intelligence), Smart City Platforms and e-Governance domains, as well as software services such as Software Quality and Security audits, IV&V, Vulnerability Assessment and Penetration Testing.

**2.3 Team Operation**

The team consists of permanent employees from BEL along with trainees and interns who were working on this project. Physical presence was required due to the highly confidential nature of the project. Our guide used to give us daily tasks and we were supposed to finish them by EOD. Being a defence enterprise, high standards and privacy were maintained for any task done by anyone. The code written by any of the team members was reviewed twice before the testing was done and eventually put into production.

Regular meetings were held to discuss the progress and mistakes, inaccuracy, etc. were rectified and appropriate corrections were made. As a whole, the team worked in a unison manner sharing the code over secured LAN ensuring a high level of security and safety.

**2.3.1 Planning**

In addition to organizing itself, one of the first steps in the execution of the static code analysis is to obtain a thorough understanding of the task at hand and the necessary requirements of the standard to be followed and the other analysis parameters. We need to understand the analysis objectives, the specifications/customer requirements, mapping targets, tools cost, and schedule. This information is provided to the team by the program manager or management team, and the team must thoroughly review these requirements.

**2.3.2 Team Building**

As teams are formed, there is a need to recognize the interpersonal dynamics that exist to make the team process effective. People assigned to the development team will represent a variety of personalities and styles. The different perspectives that the people bring to the team can enhance its vitality and creativity.

**2.4 Static Code Analysis Phases**

****

**2.4.1 Write the Code**

The first step is to write the code. The code can be a format and language and an appropriate static code analyzer should be used.

**2.4.2 Run the Static Code Analyser**

Next, we run a static code analyzer over the code written. It will check the code against predefined coding rules. These might be from a [coding standard](https://www.perforce.com/resources/qac/coding-standards). Or they might be in-house coding rules that the team has developed.

**2.4.3 Review the Results**

The static code analyzer will identify code that doesn’t comply with the coding rules. You can then review the results. There may be [false positives](https://www.perforce.com/blog/qac/what-are-false-positives-and-false-negatives-plus-examples) to dismiss. And there will be some issues that are more important to fix than others. Some tools, such as [Helix QAC](https://www.perforce.com/products/helix-qac) and [Klocwork](https://www.perforce.com/products/klocwork), will prioritize the violations for you.

**2.4.4 Fix what needs to be Fixed**

Next, you fix the issues that need to be fixed. Start with the most critical fixes. And go down the list from there.

**2.4.5 Move to Testing**

Once we have resolved issues in the code, we can move on to the next phase of development which is testing. And we can begin the process over again.

**Chapter 3**

**Task Performed/ Internship Activities**

**3.1 Learning about static Code Analysis**

Static Code Analysis (also known as Source Code Analysis) is usually performed as part of a Code Review (also known as white-box testing) and is carried out during the Implementation phase of a Security Development Lifecycle (SDL). Static Code Analysis commonly refers to the running of Static Code Analysis tools that attempt to highlight possible vulnerabilities within ‘static’ (non-running) source code by using techniques such as Taint Analysis and Data Flow Analysis.

Ideally, such tools would automatically find security flaws with a high degree of confidence that what is found is indeed a flaw. However, this is beyond the state of the art for many types of application security flaws. Thus, such tools frequently serve as aids for an analyst to help them zero in on security-relevant portions of code so they can find flaws more efficiently, rather than a tool that simply finds flaws automatically.

Some tools are starting to move into the Integrated Development Environment (IDE). For the types of problems that can be detected during the software development phase itself, this is a powerful phase within the development lifecycle to employ such tools, as it provides immediate feedback to the developer on issues they might be introducing into the code during code development itself. This immediate feedback is very useful as compared to finding vulnerabilities much later in the development cycle.

## Techniques

There are various techniques to analyse static source code for potential vulnerabilities that may be combined into one solution. These techniques are often derived from compiler technologies.

### Data Flow Analysis

Data flow analysis is used to collect run-time (dynamic) information about data in software while it is in a static state ([Wögerer, 2005](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.394.5540)).

There are three common terms used in data flow analysis, basic block (the code), Control Flow Analysis (the flow of data) and Control Flow Path (the path the data takes):

Basic block: A sequence of consecutive instructions where control enters at the beginning of a block, control leaves at the end of a block and the block cannot halt or branch out except at its end ([Wögerer, 2005](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.394.5540)).

Example PHP basic block:

$a = 0;

$b = 1;

if ($a == $b)

{ # start of block

echo “a and b are the same”;

} # end of block

else

## { # start of block

## echo “a and b are different”;

## } # end of block

## Control Flow Graph (CFG)

## An abstract graph representation of software by the use of nodes that represent basic blocks. A node in a graph represents a block; directed edges are used to represent jumps (paths) from one block to another. If a node only has an exit edge, this is known as an ‘entry’ block, if a node only has an entry edge, this is known as an ‘exit’ block ([Wögerer, 2005](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.394.5540)).

## Example Control Flow Graph; ‘node 1’ represents the entry block and ‘node 6’ represents the exit block.

## Control Flow Graph

## Taint Analysis

## Taint Analysis attempts to identify variables that have been ‘tainted’ with user-controllable input and traces them to possible vulnerable functions also known as a ‘sink’. If the tainted variable gets passed to a sink without first being sanitized it is flagged as a vulnerability.

## Some programming languages such as Perl and Ruby have Taint Checking built into them and enabled in certain situations such as accepting data via CGI.

## Lexical Analysis

## Lexical Analysis converts source code syntax into ‘tokens’ of information in an attempt to abstract the source code and make it easier to manipulate (Sotirov, 2005).

Pre-tokenised PHP source code:

## <?php $name = "Ryan"; ?>

## Post tokenised PHP source code:

## T\_OPEN\_TAG

## T\_VARIABLE = T\_CONSTANT\_ENCAPSED\_STRING;

## T\_CLOSE\_TAG

**3.2 About PyLint**

It is a Python static code analysis tool that looks for programming errors and helps to enforce a

coding standard sniffs for code smells and offers simple refactoring suggestions.

It’s highly configurable, having special pragmas to control its errors and warnings from

within your code, as well as from an extensive configuration file. It is also possible to write

your plugins for adding your checks or for extending pylint in one way or the other.

Pylint has many rules enabled by default, way too much to silence them all on a minimally sized program. It's highly configurable and handles pragmas to control it from within your code. Additionally, it is possible to write plugins to add your own checks. It's free software distributed under the GNU Public Licence.

From the authors of flake8 - pylint, checks for PEP8-like code style, some code smells and type errors. It intersects with flake8 in many regards, but doesn’t check for complexity (can be enabled through the plugin pylint.extensions.McCabe).

Pylint doesn’t just have the best range of features, but it’s also constantly maintained, making it a must-have tool for Python developers. It’s been around for 13 years, and over that time it has included features like coding standards, error detection and refactoring by detecting duplicated code.

It is easy to set up, requiring a minimal amount of configuration, but it’s fully customizable if you want it to be — by editing a file you can select which errors and conventions are most relevant to you.

**3.3 Project Requirements**

#### **Software Requirements**

* Python 3.x
* Pip
* Any Code Editor

#### **Hardware Requirements**

* A standard computer

**3.4 Installation of Pylint**

1. For python 2.7.9 and below:

a. Download the source distribution version 1.9.5 from the given link https://github.com/PyCQA/pylint/releases/tag/pylint-1.9.5

b. Extract all the files and open the terminal/ command prompt in the extracted folder

c. Run the command python setup.py install

2. For python 2.7.9 and above :

a. Open the command prompt/ terminal

b. Run the command pip install pylint

c. The user can also download the latest version from the source distribution (link: https://github.com/PyCQA/pylint) and follow the installation steps in method 1.

**3.5 Run Pylint**

1. Pylint is meant to be called from the command line. It can be used as pylint [options] modules\_or\_packages (make sure the path is proper)

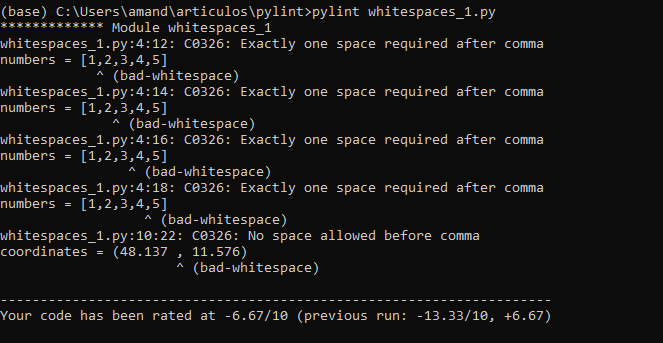
2. We can also use pylint as pylint directory/mymodule.py

3. We can also achieve parallel execution by using the following command pylint -j 4 mymodule1.py mymodule2.py mymodule3.py mymodule4.py

4. To run the pylint for the entire package use command find . -type f -name "\*.py" | xargs pylint.

5. pylint directory/mymodule.py can be also used if the directory is a python package (i.e. has an \_\_init\_\_.py file or it is an implicit namespace package) or if the “directory” is in the python path.

**3.6 Output after running Pylint**

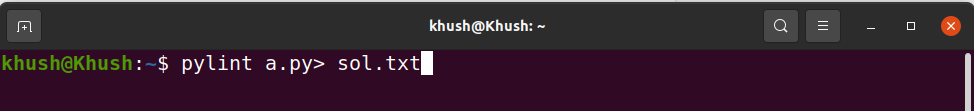


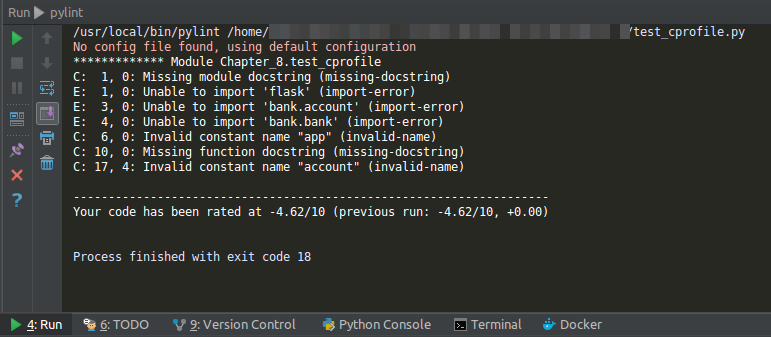
The above image is the output on running a pylint on a single module.

We can get the output in the same format of various commands mentioned in the “How to Run” section.

Note: To save the output in a different file we can use the redirection operator “>” common for both linux based os and windows.

For example:





● We see that after running the command in figure 2 we get a file saved (figure 3) in the same package in the format that we need it in (.json, .html, .txt, etc.).

● The output format and view can also be changed into the format that we need it in. ● More information on the same can be seen over here

https://pylint.pycqa.org/en/latest/user\_guide/output.html

● Integrating pylint with various editors and Ide can also be done.

**Chapter 4**

**Specific Outcomes**

**4.1 Technical Outcomes**

During my internship program, I understood the importance of static code analysis and why it is important to maintain a standard of code across an organization. I learned about different static code analysers and why it is important to have a globally accepted static code analyser in place.

I also learned about the practical implementation of HTML, CSS and JS by making a website during the internship period. I also saw how radars were tested and how the nation’s defence system is in place and how important the testing for the same.

**4.2 Non-Technical Outcomes**

The Internship program helped me gain self-confidence in using the tool and also helped me enhance my communication skills. I was also a Team lead for the team of interns which helped me improve my leadership skills, The project coordinators were very helpful and guided me to understand the entire process. This made me learn easily and effectively.

### **Communication Skills:**

This Internship program helped me to communicate with the project managers, project co-coordinators, and team members, which made me gain confidence in my communication skills.

### 

### **Time Management:**

The Internship program helped me in maintaining time and completing tasks in a given time slot. Periodical interaction with the manager was undertaken so that any changes in the project would be carried out immediately. The assigned tasks were completed on time. This made me learn time management

### **Leadership Skills:**

This Internship program helped me to learn how to lead a team and manage the working pressure and even how to make our communication more effective. The project managers, project co-coordinators, and team members made me gain confidence in my leadership skills.



**Chapter 4**

**Conclusion**

Throughout the duration of the internship, I have learned daily and developed both technical and interpersonal skills. I also learned many core concepts which helped me get a strong hold on the basics of many technical skills.

The most important thing I learned is that "Sticking to the basics" is the key to the professional world and not trying too many fancy things which may complicate a simple task.

Professionalism and Code of Conduct is the other thing that I learned throughout the course of the internship which includes (a way of talking to colleagues, sitting in a professional environment, a professional way of coding, etc.)

During the internship, the benefit I achieved is to test out a job, employment setting, management style, and other aspects of the workplace.

**References**

**[1] https://www.bel-india.in/Default.aspx**

**[2] https://luminousmen.com/post/python-static-analysis-tools**

**[3] https://www.w3schools.com/**

**[4] https://www.perforce.com/blog/sca/what-static-**

**analysis#:~:text=Static%20code%20analysis%20is%20a,along%20with%20source%20code%20analysis.**

**[5] BEL Internal Resources**