Software Support to Memory Generation and Monitoring Platform

Submitted By

Priyanka Charoliya

15MCEC06

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2017

Software Support to Memory Generation and Monitoring Platform

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

Submitted By

Priyanka Charoliya

(15MCEC06)

Guided By

Internal Guide: Asst. Prof. Usha Patel

External Guide: Mr.Vivek Garg

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2017

Certificate

This is to certify that the major project entitled **"Software Support to Memory Generation and Monitoring Platform"** submitted by **Priyanka Charoliya (Roll No: 15MCEC06)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering of Nirma University, Ahmedabad, is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Usha Patel
Guide & Assistant Professor,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Prof. Priyanka Sharma
Associate Professor,
Coordinator M.Tech - CSE
Institute of Technology,
Nirma University, Ahmedabad

Dr. Sanjay Garg
Professor and Head,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr Alka Mahajan
Director,
Institute of Technology,
Nirma University, Ahmedabad

Statement of Originality

I, **Priyanka Charoliya, 15MCEC06**, give undertaking that the Major Project entitled "**Software Support to Memory Generation and Monitoring Platform**" submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Signature of Student
Date: 17 May, 2017
Place: Ahmedabad

Endorsed by
Asst. Prof. Usha Patel
(Signature of Guide)

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Mrs. Usha Patel**, Assistant Professor, Computer Science Department, Institute of Technology, Nirma University, Ahmedabad for her valuable guidance and continual encouragement throughout this work. The appreciation and continual support she has imparted has been a great motivation to me in reaching a higher goal. Her guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

It gives me an immense pleasure to thank **Dr. Sanjay Garg**, Hon'ble Head of Computer Science and Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr Alka Mahajan**, Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation she has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

- Priyanka Charoliya

15MCEC06

Abstract

Customers put request for products. to deliver product as per customer's requirement, we have to resolve failures that arise during delivery. For that we will monitor jobs submitted on LSF (Load Sharing Facility) platform. Tasks that are performed during monitoring are discussed here. This monitoring tasks are automated by executing scripts written in scripting languages. To track these failures in future, we will introduce a tool - Codex. Working of this tool is optimized by automating some tasks performed by Codex. Further we analysed the failure occurred monthly and also in terms of types of failure in year 2015-2017. Moreover the efforts to handle failures monthly and also in terms of types of failures.

Abbreviations

TCL	Tool Command Language
VNC	Virtual Network Computer
LSF	Load Sharing Facility
SWIG	Simplified Wrapper and Interface Generator

Contents

Certificate iii

Statement of Originality iv

Acknowledgements v

Abstract vi

Abbreviations vii

List of Figures x

1 Introduction 1
- 1.1 Background . . . 1
  - 1.1.1 Library Views . . . 1
- 1.2 LSF - Load Sharing Facility . . . 2
  - 1.2.1 Need for LSF . . . 3
  - 1.2.2 Job Submission and Control . . . 3
- 1.3 Motivation . . . 5
- 1.4 Objective . . . 5
- 1.5 Scope of Work . . . 5
- 1.6 Tools and Technology . . . 6

2 Literature Survey 7
- 2.1 Version Control . . . 7
- 2.2 Code Review . . . 7
- 2.3 Continuous Integration . . . 8

3 Software Support to Memory Generation 9
- 3.1 Product Indexing . . . 9
- 3.2 Hard-link for Directory . . . 11

4 Tool: Codex 13
- 4.1 Automation in Codex . . . 14
  - 4.1.1 Automatic Codex Creation . . . 14
  - 4.1.2 Automatic Fill Description Field . . . 14
- 4.2 Updating in Codex . . . 15
  - 4.2.1 Update in Codex Creation . . . 15
  - 4.2.2 Update in Flow for Calling Scripts . . . 15

5 Tool to Access C++ Functions in TCL **16**
5.1 Requirement . . . 16
5.2 Existing Tools . . . 16
5.3 Solution . . . 17

6 Monitoring **18**
6.1 Implementation . . . 18
6.2 Steps for Job Monitoring . . . 22
6.2.1 Failed Generations . . . 22
6.2.2 Tanked Generations . . . 24
6.3 Automation . . . 27
6.3.1 Automatic Codex Creation for Generations . . . 27
6.3.2 Failed Generation Categorization According to Responsible Team 27
6.3.3 Restart Tanked Generation . . . 27
6.4 Auto-completion of Script Options . . . 29
6.4.1 Auto-completion of Script Options . . . 29
6.4.2 Standard Completion . . . 29
6.4.3 Proposed Completion . . . 30

7 Analysis **31**

8 Conclusion and Future Work **33**

Bibliography **34**

List of Figures

1.1 Library Views 1

3.1 Script to find products, not found in prod 10
3.2 Hard-link 12

6.1 Output of script - GetWorkingDir 19
6.2 Output of script - DailyAlert 21
6.3 Failure occurrences 24
6.4 Failure Types 25
6.5 Flow of Monitoring Process 26
6.6 Variable Name Completion 29
6.7 Username Completion 29
6.8 Filename and Directory Completion 30
6.9 Script Option Completion 30

7.1 Failure Occurrences in Year 2015-2017 31
7.2 Failure of Types in Year 2015-2017 32
7.3 Efforts to handle Failures in Year 2015-2017 32
7.4 Efforts to handle Failures in terms of Types of Failures in Year 2015-2017 32

Chapter 1

Introduction

1.1 Background

Cells are components performing basic functions (Boolean functions like AND, OR, etc) [3]. The collection of these cells is called Standard cell Library. Another definition of library is, it is a consolidated data that is used in design of a system on chip. It has various views. These views are helpful in designing of a chip. A cell is given as views. various tool uses each view in designing of a chip.

1.1.1 Library Views

View is a specific representation of a cell. They are used in various tools in design flow of memory generation. Classification of views are shown in figure 1.1.

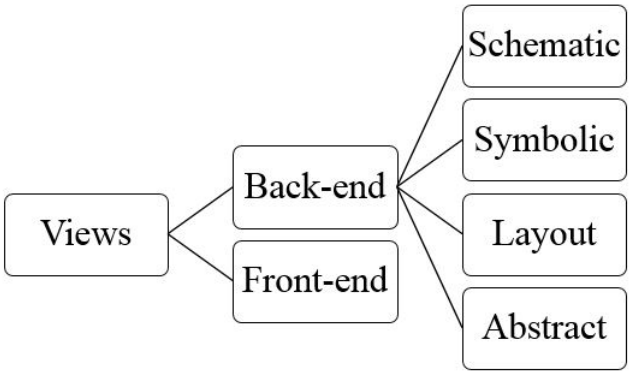

Figure 1.1: Library Views

Back-end views are concerned with the physical design of a cell and front-end views are concerned with the timings/modeling of the Cell. Here we are only concerned with

the back-end views.

Back-end Views

- **Symbol view:** What we will see on chip is considered in this view. Pictorial representation is symbol of function and text representation is .slib file. The symbol's shape demonstrates the function of cell. In design documentation, labels given in the symbol are used.
- **Schematic view:** It is simplified representation of electric circuit. Cells are represented at transistor level. Include pins, symbol graphics, labels. Text representation of schematic is .cdl (Circuit Description Language) file [3].
- **Layout view:** It is physical representation of electric circuit. Physical representation is made in terms of geometric planar shapes that corresponds to semiconductor layers which makes the integrated circuit components. Text representation of schematic is .gds (Graphical Design System) file, which is platform independent [3].
- **Abstract view:** Abstract view provides information related to the signal and power pin layers that are running in the layout view. ASCII representation of this view is .lef (Library Exchange Format) file, which includes library information for design of a chip [3].

1.2 LSF - Load Sharing Facility

Load Sharing Facility is a load balancing application by IBM. To optimize the run time of jobs, it combines many servers. [2] Platform LSF (Load Sharing Facility) is a distributed resource management product suit that [6]:

- Connects computers in Grid or Cluster.
- Monitors load of systems.
- Distributes, schedules and balances workload.
- Controls access and load by policies

- Analyzes the workload
- Provides transparent access to all available resources

1.2.1 Need for LSF

In distributed environment without LSF, users will observe lack of resources because of resources and workload are not properly allocated. There is no centralized control of resources in absence of LSF. User have to find manually, which node can run his task. With LSF, user doesn't have to worry about which node will run his task. It is LSF's responsibility to find the best node to run that task.

1.2.2 Job Submission and Control

We have setup LSF compute farm, in such a way that it has multiple clusters. To know the version of LSF and name of clusters available, we can use below mentioned command.

lsid

To get information about local cluster of LSF such as number of servers, number of hosts, account name of admin, cluster status, etc, below mentioned command is used.

lsclusters

Each cluster has different types of queues. We can get the information of these queues, such as name of queue, priority of that queue, status of it, etc using the command given below.

bqueues

Job is submitted to one of these queue using *bsub* command. LSF decides on which queue this job is to be submitted according to:

- User access: If user is not allowed to submit jobs on specified queues, those queues are excluded.

- Host: If user explicitly specify on which host job is to be run, then all other hosts are excluded.
- Resources: If resources requested by job is not within the limits of queues, then that queues are excluded.

Job remains pending until all conditions for its execution are met. When LSF all conditions are met, then the job is considered for dispatch in next dispatch run. After dispatching job, LSF places the job on the best available host and that host runs the job. [2] There are other commands that are used for job control. Such as [6]:

- Modify Pending/Running Jobs: *bmod*
- Kill Jobs: *bkill*
- Suspend Jobs: *bstop*
- Resume Jobs: *bresume*

To get all jobs on LSF, command mentioned below is used.

bjobs

We can filter jobs as per our need using the following options:

- *-p*: Prints jobs that are pending , with the reasons which made them not to be dispatched.
- *-r*: Gives running jobs.
- *-s*: Prints jobs that are suspended , with the reason which made them to come to be suspended.
- *-g*: Prints information of jobs that are related to specified job group.
- *-j*: Prints information of the job that is having specified job name.
- *-m*: Displays jobs dispatched to the specified hosts, host group, cluster.
- *-o*: Sets the customized output format.
- *-P*: Prints jobs that are belonging to the project specified.

- *-q*: Prints jobs in queue which is specified.
- *-u*: Prints jobs submitted by user or user group that is specified.

1.3 Motivation

A product is a set of documents and files that is used in commercial tools or tools written in many different programming languages. Product specifications are given by a customer as per his requirements. As we receive request from customer, we start developing product. When product is ready, it is our responsibility to deliver it to a customer as he require. While delivering product to a customer, before delivery failures can arise. These failure must be resolved. This process of resolving failures is called monitoring. During monitoring we use tool Codex to record each and every failure. Moreover user-friendly environment should be provided for ease of work.

1.4 Objective

The objective of the project is to automate and optimize working of tool Codex in terms of execution time and scalability and to implement monitoring tasks. Furthermore it is also required to provided user-friendly environment.

1.5 Scope of Work

A tool for Application Lifecycle Management is introduced. Various tasks performed on this tool are automated. Scripts running in back-end of this tool are updated to optimize working of tool. In addition to this, job monitoring process is to be carried out for product generations that are failed while delivering to the customer and appropriate steps are to be initiated. Further this monitoring process is also automated by running scripts every interval of time. One tool is introduced, using that, we can access C++ functions can be accessed from TCL scripts.

1.6 Tools and Technology

- **Operating System:**
  - RedHat Linux
- **Languages:**
  - **Shell Script:** Shell is a command language interpreter. It executes commands that it has got from the standard input device (keyboard) or from a file. Shell script is a set of shell commands whose execution ow can be changed.
  - **TCL:** TCL is an abbreviation for Tool Command Language. It is a scripting language. It was designed with the goal of being very simple but powerful. It is mainly used for text processing and file handling. When we need data structures, shell script do not provide them. In this case we can use TCL.
- **Tool:**
  - **VNC:** VNC is an abbreviation for Virtual Network Computer. It is a desktop sharing system. It follows client-server model. So, one or more clients can connect to one server at a same time. VNC is mainly used to access files at remote locations.

Chapter 2

Literature Survey

2.1 Version Control

Version control helps a software team to maintain changes in source code till date. Version control keeps track of each and every modification done in the code. without version control, software development is risky, as not having any backups. If you have the complete history, it enables you to look back to previous versions that will help in root cause analysis for finding bugs.

There are many Version Control Systems available as open source.

Problem with existing tool:

- Restrictive design, which limits the capacity of an Agile-oriented management team.
- Many of them are not easy to setup or maintain.

2.2 Code Review

While reviewing code, other developers can review a proposed change. Changes can be suggested and updated by every contributor. Once all contributors accept the change, it is merged into the base code.

Advantages

- Early error detection.

- Reviewer can spot logical flaws before code is merged into base code.
- Allows the team to identify any violations with the team code standards early in the process.

There are many open source Code Review Systems, such as Gerrit.

Problem with existing tool:

- Post-merge review is impossible - Once a change is merged, it's sacred. we cannot make it "unmerged", and leave pending for a new merge after it gets fixed.
- One commit at a time workflow - Even if multiple dependent commits are pushed to the repository, a need to update one of them breaks a whole chain of dependencies between them.
- The single commit based submission - Changes can be committed by author only.

2.3 Continuous Integration

In continuous integration, isolated changes are tested immediately. Before changes are added to a base code, they are reported on. Provide rapid feedback is the goal for continuous integration. So that, if any defect found in the code, it can be corrected as soon as it is identified.

There are many tools available to automate continuous integration tasks testing and building documents, such as Jenkins.

Problem with existing tool:

- A lot of problems in the tracking changes made by the various members of the development team.
- Securing access to the machine and regularly updating to patch the latest OpenSSL vulnerabilities.

Chapter 3

Software Support to Memory Generation

3.1 Product Indexing

To understand product indexing we need to know about some keywords such as PATH, prod, .prod. PATH is environment variable which is set by user. PATH is set with the path of central repository from that user can get required product package using product indexing [6]. prod is a indexing file which contains following information about products.

- Product name
- Product version
- Product Attributes
  - "+": used for tools
  - "-": used for libraries
  - ".": used for hidden products
- Supported platform
- Product type
- Date of product creation
- Path of product

.prod is a file that is written by user and stored where user want product to be installed. .prod contains following fields:

- Product name: Product that user want to install.
- Product version

Now, it is required to understand how product indexing is done. User set environment variable PATH with the path of central repository. Then user write product name and version in .prod file. User can write one or more product in .prod.

When user run command for installation on terminal, the product indexing is done using following procedure.

```
if .prod file exists in installation directory
    foreach product in .prod
        foreach path mentioned in environment variable PATH
            check for prod file
            if product information found in prod
                get path of product package
                get product package from that path
                break
            else
                print "Product not found"
else
    give error
```

```
createUcdprod out /sw/unicad/
awk '{print $NF}' out | grep /sw/unicad/ | sort -u > fout
ls /sw/unicad/*/*.ptbl | cut -d/ -f 1-4 | sort -u > 1
ls /sw/unicad/*/*/*.ptbl | cut -d/ -f 1-5 | awk -F "/" '{if ($NF ~ /[0-9]/){print $1"/"$2"/"$3"/"$4"/"$5} else {print $1"/"$2"/"$3"/"$4}}' | sort -u >
2
ls /sw/unicad/*/*/*/*.ptbl | cut -d/ -f 1-5 | awk -F "/" '{if ($NF ~ /^[0-9]/){print $1"/"$2"/"$3"/"$4"/"$5} else {print $1"/"$2"/"$3"/"$4}}' | sort -u
 > 3
grep -vf fout 1 | sort -u > f1
grep -vf fout 2 | sort -u >> f1
grep -vf fout 3 | sort -u >> f1
```

Figure 3.1: Script to find products, not found in prod

- **Previous Scenario:** Some of the products were not found in prod.
- **Problem:** When more than one versions of the same product are stored in same directory, only one version was found in prod.

- **Solution:** Change in script that is used for product indexing.
- **Effect:** All versions of all products are be found in prod, so that customer can get them when they need.

3.2 Hard-link for Directory

Before creating hard-link we need to understand what is link, what are the different types of link, what will they do, etc. In UNIX links are pointers, that points to a file or a directory. Creation of links is a kind of shortcuts for accessing a file. There are two types of link, soft-link and hard-link.

Soft-link

A soft link is a file that points to another file in the file system. It is like shortcut in Windows. Soft link does not contain the data in the target file. Also, if we delete a target file, links to that file will be unusable. To create soft-link, execute the following command:

ln -s file/directory link

Hard-link

A hard link is a name that is given to a file. Create a number of different names that all refer to the same content, is also possible. Commands executed on any of these names will then perform operation on the same file. To create hard-link, execute the following command:

ln file link

This link is not a separate copy of the file, but it is a different name for the same contents as the file. Any changes made in file will be visible in link. The concept of hard-link is shown in figure 3.2.

We can create hard-link for files. To create hard-link for directory, we have to write script.

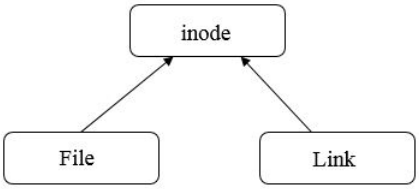

Figure 3.2: Hard-link

- **Previous Scenario:** A .tar file is loaded in user area and then it is extracted.
- **Problem:** If a .tar file is loaded and then extracted in user area, it will require more space.
- **Solution:** Instead of extracting .tar file in user area we can extract it at source location and create hard-link of that extracted directory in user area.
- **Effect:** Less time is required to create hard-link compared to loading a .tar file and then extracting it when it is needed. Another advantage is extracted .tar file can be reused by other user also.

The advantage of using hard link is after deleting file hard links retain the file contents. So that if for any reason, product is deleted from central repository, users can use their copy of product.

Here one problem can arise, that is if more than one user want to access the same extracted directory, inconsistency may occur. To solve this problem, locking is used. When one user is accessing the directory, other user can not get access to that.

Chapter 4

Tool: Codex

Codex is a web-based Application Lifecycle Management software. Codex's high customization capability enables you to tailor each tool to your processes and business requirements.

With Codex, you will be able to [6]:

- Manage the entire lifespan of your software projects with one single suite.
- Manage source code and collaborate through Git, Gerrit.
- Track bugs, requirements, tasks, incidents, etc.
- Manage agile, traditional and hybrid methodologies within the same platform.
- Set up new projects in a few clicks, defining the project team and set access right in different area in a detailed way.
- Leverage integration with leading open source tools as Git, Gerrit or Jenkins.
- Share documents and releases.

Codex will place developer, product owner, project manager and customer together. The additional features that Codex provides are:

- Automatically trigger continuous integration.
- Trace documentation, easily plan work, review and test code, manage issues.

- Share your releases and deliverable along with related issues and maintain permissions on each files.

Moreover, instead of maintaining three different tools for version control, code review, continuous integration and integrate them, we can do all of them in one. This will reduce our cost to purchase licence for different tools for three different tasks. It will also reduce our work to maintain those tools.

4.1 Automation in Codex

The task of creation of Codex can be automated for all details except the corrective action [6].

4.1.1 Automatic Codex Creation

- **Previous Scenario:** Codex entry was created manually.
- **Problem:** More time is wasted in manually creating codex entry.
- **Solution:** Creation of Codex entry can be automated by executing script.

4.1.2 Automatic Fill Description Field

- **Previous Scenario:** Manually write value for Description field while executing script for Codex creation.
- **Problem:** Users have to do analysis to find reason for generation failures manually and then write in description field of codex entry.
- **Solution:** Write a script that read files that contains error and extract this error and call this script in script to create Codex and fill found error in description field.

The algorithm implemented for above automation is mentioned below.

Algorithm:

foreach log file in working directory

    foreach line in log file

        if line contains error

            append line to Description field

4.2 Updating in Codex

Update in scripts running in back-end of tool to optimize working of tool.

4.2.1 Update in Codex Creation

- **Previous Scenario:** Script to create Codex entry is using a function that was reading files in generation working directory to fill details for fields while executing command for creating Codex entry.
- **Problem:** The function used was checking if files for fields passed while executing command for creating Codex entry are existing or not. If they are existing then it was reading those files for each and every fields that are passed and return their value to calling script.
- **Solution:** Remove use of this function and make change in script to fill some of the fields automatically.

4.2.2 Update in Flow for Calling Scripts

- **Previous Scenario:** There were different scripts for different tasks performed on Codex.
- **Problem:** More space is required. Difficult to maintain and debug each and every script as separate file.
- **Solution:** Write a script that contains different scripts for different tasks as a function. So that programmer have to debug only this script only.

Chapter 5

Tool to Access C++ Functions in TCL

5.1 Requirement

We have many functions that are written in C++, and we wanted to use those functions in our TCL script. As we didn't want to write the whole function again in TCL, it was desired that if we could access those function in our script without changing them.

Naive method to access function written in C++ in TCL, is to write wrapper code. But this is equivalent to write the whole function in TCL.

5.2 Existing Tools

There are some tools available, such as C++/Tcl, SWIG, etc. By using these tool we do not have to write wrapper code for each functions that we want to access in TCL. These tools do that for us. But they have drawbacks in one way or another as mentioned below.

- We have to append some code in source file if we use C++/Tcl [5].
- We need to write interface file for SWIG.
- Most of them do not support all features of C++.
- We may need to eliminate main function from source file.
- Some of them do not work in multi-threaded environment.

5.3 Solution

We have developed one tool that will write interface files for us, that we can use in SWIG. SWIG will generate wrapper code of functions written in C++, that we want to access in TCL [1]. Using these wrapper code we can generate shared object(.so) file that we can directly load in TCL. After that we can access function written in C++ just by using name of those functions.

For that we have used Flex and Bison as lexical analyser and parser respectively. Flex is a tool that generates scanners, that recognizes lexical patterns in file. The flex program reads the input files, then it gives a C file as an output. To produce an executable, this output file can be compiled and linked with the flex runtime library. It analyzes input for regular expressions occurrences, when the executable is run. if it gets particular regular expression, then it executes related C code. The parser reads a series of tokens and tries to determine the grammatical structure [4]. The reasons why we use Flex and Bison are:

- Flex and Bison are open source.
- They generates a parser that is faster.
- Updating in source files is a much easier than updating custom parser code.
- They have mechanisms for error handling and recovery.

As Future work we want that we remove the use of SWIG and build stand-alone product.

Chapter 6

Monitoring

6.1 Implementation

The following scripts are executed as terminal command while monitoring jobs.

- **GetWorkingDir:** This script is used to get working directories from compute-farm based on various parameters. Options accepted by the scripts are:
  - **-w** : Path(s): 'path1 : path2'.
  - **-r** : Requester name.
  - **-s** : Status: 'stat1/stat2'.
  - **-l** : Library name.
  - **-cfg** : Configuration name.
  - **-p** : Project name.
  - **-cat** : Category name.
  - **-j** : Lsf main job id.
  - **-o** : Obsolete directory yes/no.
  - **-r_team** : Team name.
  - **-orph** : Directories without LSF jobs.
  - **-started_after** : Date to start searching
  - **-started_before** : Date to stop searching.
  - **-dm_id** : Designsync ID.

- **-display** : Display type. Default is 'plain'.
- **-lr** : Display latest for each designSync request.
- **-help/-h/-u/-U** : Display the script usage.
- **-gui** : Graphical user interface.

The following figure 6.1 shows the sample output of script.

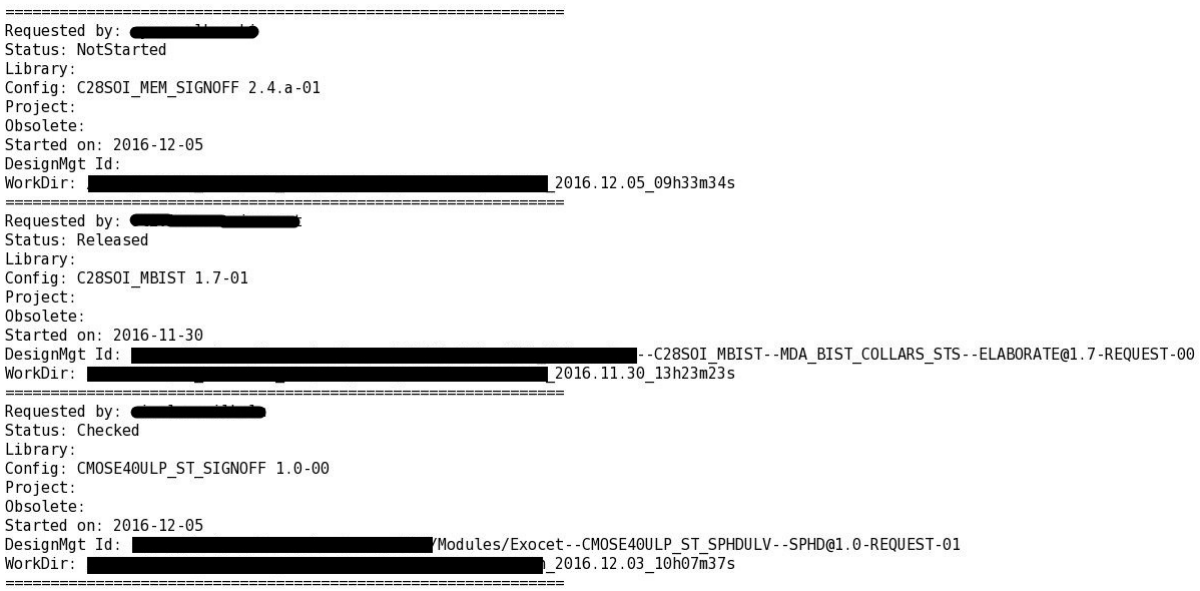

Figure 6.1: Output of script - GetWorkingDir

- **IsLatestGeneration:** The script will check whether a generation is the latest that is to be launched or not. If it is not latest generation then it will display the working directory of latest generation. Option accepted by this script:
  - **-w** : Generation Working directory.
  - **-help/-h/-u/-U** : Display the script usage.
  - **-gui** : Graphical user interface.
- **RemoveWorkDir:** This script will check if generation having specified working directory is released or marked as obsolete then it will delete the working directory. The reason behind it is to make room for other generations. Option accepted by this script:
  - **-w** : Generation Working directory.
  - **-help/-h/-u/-U** : Display the script usage.

- **-gui** : Graphical user interface.

- **GetTankedJobs:** This script will display all generations that are not progressing for hours. Options accepted by the scripts are:
  - **-main** : Time main-jobs RUNNING (hours).
  - **-sub** : Time Sub-Jobs RUNNING (hours).
  - **-pend** : Time all-Jobs PENDING (hours).
  - **-jobGroupBase** : jobGroupBase.
  - **-help/-h/-u/-U** : Display the script usage.
  - **-gui** : Graphical user interface.

- **DailyAlert:** This script will send mail about failed or tanked generation to the team members that are responsible for monitoring task on daily basis.

  The script runs at specified intervals of time. It generates results and sends them as mail to responsible team members. Scheduling of script is being done by Crontab. Cron is software utility that schedule jobs. It is available for unix based environments. It is useful in running jobs at specified intervals of time. Crontab (or Cron table) is a configuration file that determines shell commands that are to be run at specified intervals of time.

  Options accepted by the script are::
  - **-mail** : No screen display, send mail instead.
  - **-recipients** : Display recipients currently registered.
  - **-repository** : User repository.
  - **-help/-h/-u/-U** : Display the script usage.
  - **-gui** : Graphical user interface.

  Screenshot of mail is shown in following figure 6.2.

- **FindFailed:** This script is used to identify if there is any error in a given generation directory. Option accepted by this script:
  - **-w** : Generation Working directory.

Figure 6.2: Output of script - DailyAlert

- **-help/-h/-u/-U** : Display the script usage.
- **-gui** : Graphical user interface.

- **CreateCodex:** This script creates Codex entry for tracking failures. Options accepted by the script are:
  - **-workDir** : list of work directory.
  - **-cmd** : Codex Command to execute. Default is 'add'.
  - **-tracker** : Codex Tracker to use.
  - **-user** : Codex UserName.
  - **-pf** : Codex password file.
  - **-id** : Codex Id to use for update operation.
  - **-site** : Site to be used prod/qa/.
  - **-status** : Tracker Status.
  - **-stepName** : Failure Step Names.
  - **-description** : Failure Description.
  - **-correctiveAction** : Failure Corrective Action.
  - **-failure_type** : Failure Type id.

- **-artifact_id** : Codex to update.
- **-summary** : Failure Summary.
- **-help/-h/-u/-U** : Display the script usage.
- **-gui** : Graphical user interface.

- **RestartGeneration:** This script is used to restart a finished/non-running generation. Options that are accepted by the script:
  - **-workingDirectories** : Directory(ies) as 'path1' 'path2'
  - **-jobGroupBase** : jobGroupBase.
  - **-help/-h/-u/-U** : Display the script usage.
  - **-gui** : Graphical user interface.

- **SetObsolete:** This script is used to make generation obsolete when it is not a latest run. Options that are accepted by the script:
  - **-w** : Generation Working directory.
  - **-help/-h/-u/-U** : Display the script usage.
  - **-gui** : Graphical user interface.

6.2 Steps for Job Monitoring

6.2.1 Failed Generations

1. Check mail or run command GetWorkingDir -s Failed for failed to get all failed generations.
2. Go to working directory of failed generation and touch Your.Name.Is.Working.There file.
3. Execute command FindFailed to see the list of log files of the failed steps.
4. Investigate inside the log files of the failed generations to see the cause of failure.

5. Once the failure is identified, the next target is to track this failure occurrence. If this is a known issue with similar previous existing Codex artifact, then we need to just create a new Codex artifact for this failure occurrence capturing the required details as mentioned below. If this is a new failure with no recorded occurrence existing, then we have to create Codex failure type artifact and Codex occurrence type artifact. Details to be populated in Codex occurrence artifact are:

   - **Failure summary:** Single line entry containing ¡workingDir¿ and the error message.
   - **Failure type:** Codex failure type artifact id for this type of failures
   - **Step name:** Select the name of the step which you see failure in.
   - **Working Directory:** Complete path of working directory of the failed generation in question.
   - **Project name:** Name of the project from which this generation was launched.
   - **Requestor name:** Name of the person who had launched (requested) this generation.
   - **Status:** Select from open or close depending upon the resolution.
   - **Severity:** usually used option is error.
   - **Assigned to:** Name of the person who is working on this failed generation being reported.
   - **Description:** This contains 2 parts:
     (a) **Error:** List of failed steps, error message as seen from the compute farm.
     (b) **Analysis:** The analysis for actual reason of failure.
   - **Corrective Action:** The action taken for this kind of failures to resolve the issue if possible or to report to the right people responsible if the issue cannot be resolved by monitoring team.
   - Once the reason for failure and the corrective actions have been identified, then one of the following actions have to be performed:
     – If the issue can be resolved by monitoring team, then:

* If this is the latest launch for this library, then perform the corrective action and restart the generation to let it continue and finish.
* If this is not the latest launch of this library, then make this generation obsolete to indicate that this is not the latest generation and not to be processed with failures.

– If the issue cannot be resolved by monitoring team, then report the issue to the right people responsible.

Screenshots of screen displaying failure occurrences and failure types are shown in figure 6.3 and 6.4 respectively.

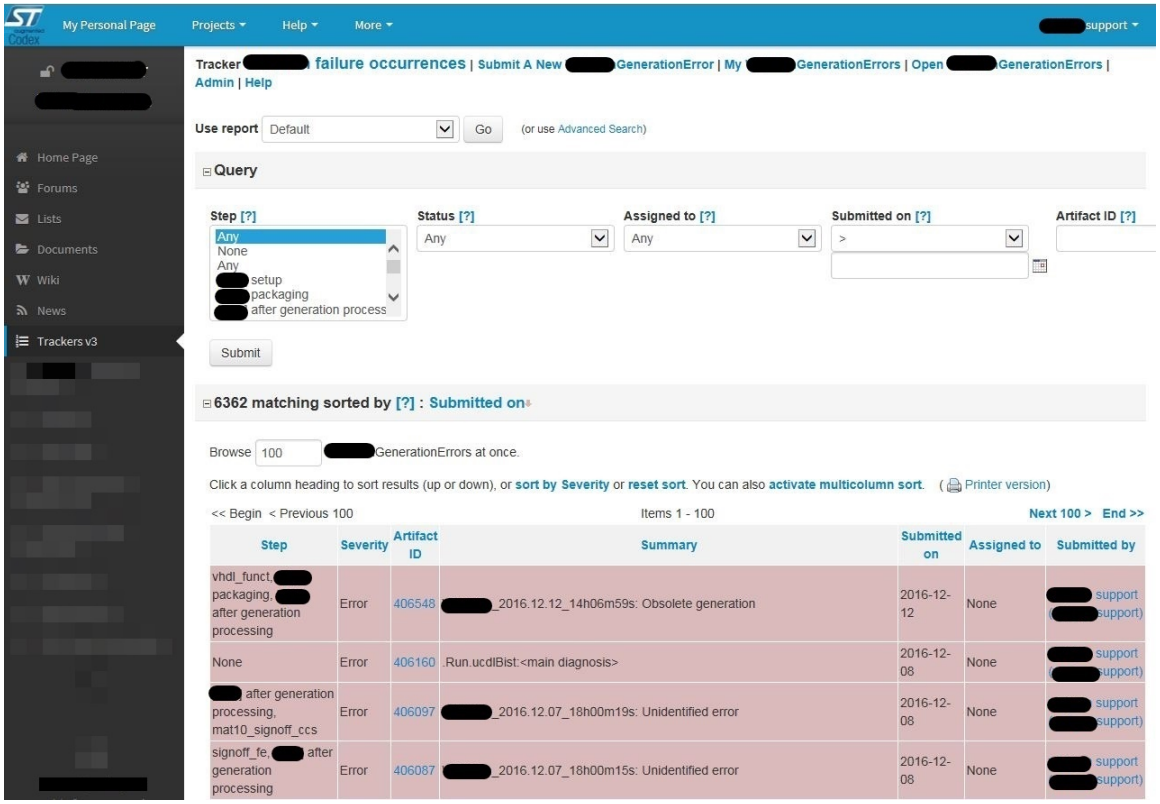

Figure 6.3: Failure occurrences

6.2.2 Tanked Generations

1. For each tanked job, go to the working directory for the jobs.

2. For each running task, check the log file for when was the last update to file done and that there is no error message inside the log file. If the log files have failures,

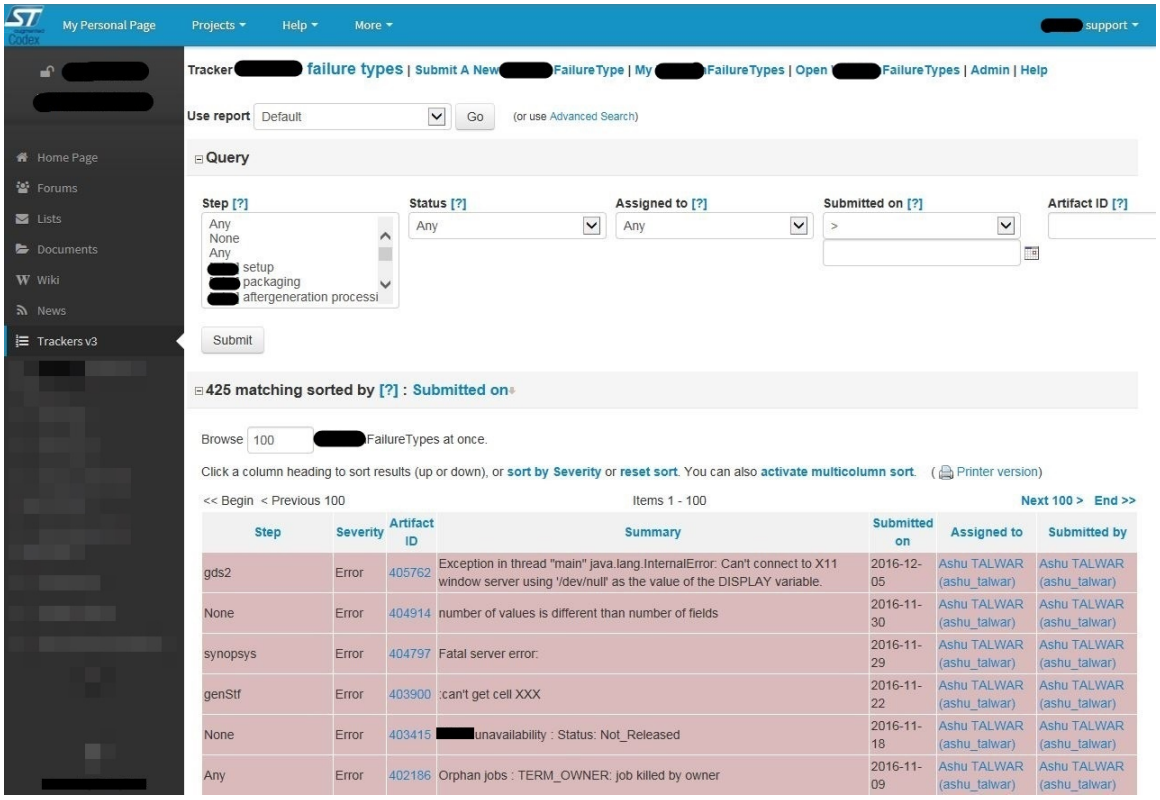

Figure 6.4: Failure Types

then cancel the generation and debug it. If the files were modified less than 12 hours, then leave the generation (if it has no errors) running.

3. For each tanked generation If there is no error inside the log files of sub-tasks and the last updates were made more than 24 hours ago, do the following procedure:

   - If this is the first instance of tanked generation, then create a Codex (occurrence) touch Your.Name.Is.Working.There file and restart the generation.
   - If this library has already been restarted earlier, then cancel the generation.

The whole monitoring process flow chart is shown in figure 6.5 [6].

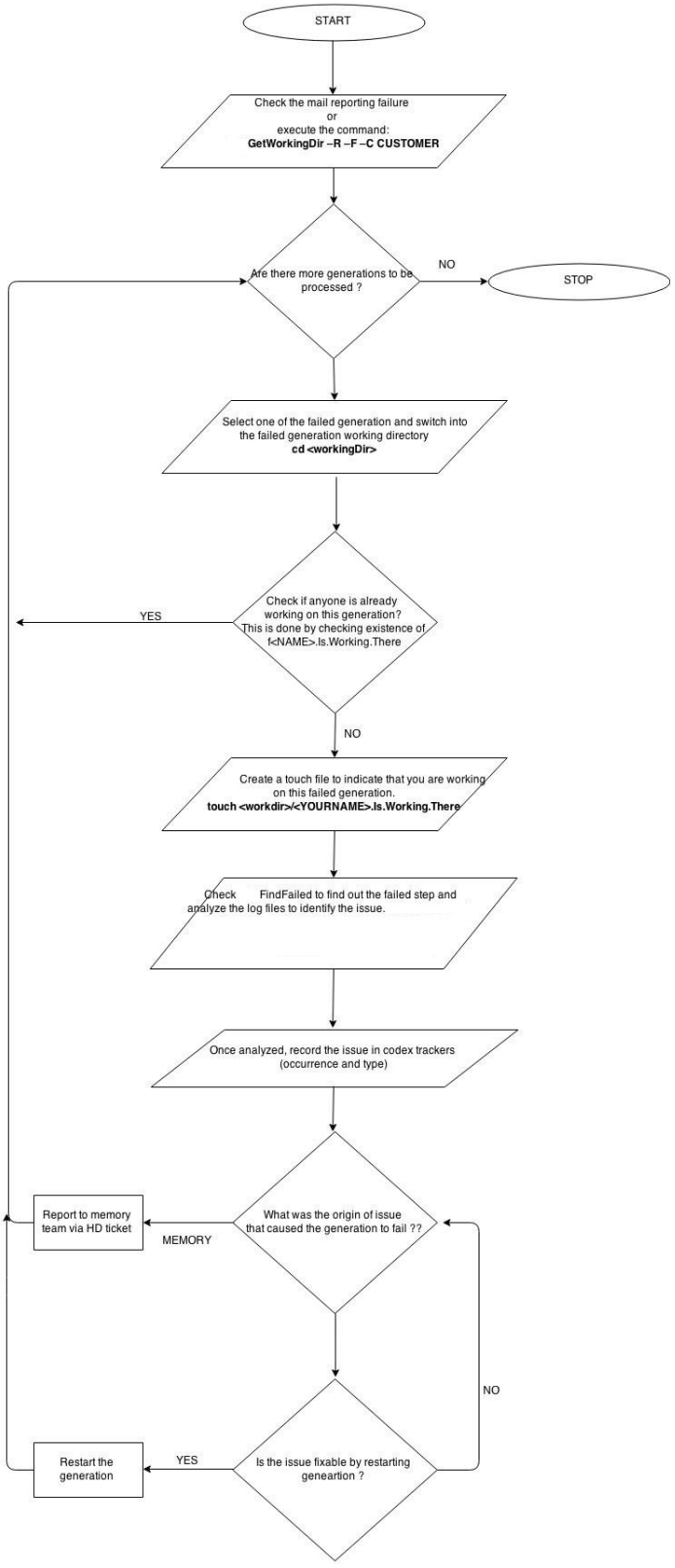

Figure 6.5: Flow of Monitoring Process

6.3 Automation

6.3.1 Automatic Codex Creation for Generations

- **Previous Scenario:** No Codex entry was automatically created for failed generations. We need to execute script for that.
- **Problem:** Users have to create Codex entry by executing script for some known failures occurring many times.
- **Solution:** For some known type of failures, we can write a script that automatically find the error and create Codex entry for that failed generation.

6.3.2 Failed Generation Categorization According to Responsible Team

- **Previous Scenario:** Mail about all failed generation is sent to the team members that are responsible for monitoring.
- **Problem:** Team members responsible for monitoring have to send mail to initiator team about generation failed due to fatal error.
- **Solution:** Check for failed generation and automatically send mail to responsible team.

Algorithm:

foreach failed generation

    read details

    find responsible team

    run mail command to send e-mail to corresponding team members

6.3.3 Restart Tanked Generation

- **Previous Scenario:** Manually checking if tanked generation is already restarted or not, and then take action according to that.

- **Problem:** Team member have to check every interval of time for tanked generation.
- **Solution:** Write a script that automatically searches for tanked generation every interval of time and checks that if generation is already restarted or not and then takes action according to that as per steps for monitoring for tanked generations.

Algorithm:

```
foreach running jobs
    find job name
    if job name contains cellData or gds
        if job is running for more than specified amount of time
            find main job from job group
            add main job ID and working directory in array
foreach job ID and working directory in array
    read log files from working directory
    foreach log file
        read log file line by line
        if generation is already restarted
            set flag
    if flag is set
        get design management ID for that working directory
        cancel generation
        restart generation
    else
        find subjobs of job
        foreach subjobs
            get jobname
            if job name contains cellData or gds
                get job ID and kill job
```

6.4 Auto-completion of Script Options

6.4.1 Auto-completion of Script Options

- **Previous Scenario:** We have to write whole script name, options and their possible values to pass as an argument to run that script.
- **Problem:** We have to remember all script options and their values type or possible values.
- **Solution:** Write a script that auto complete the script name and give list of options by pressing tab after script name and give possible values by pressing tab after relevant option.

6.4.2 Standard Completion

Linux provides standard completion mentioned below:

- **Variable name completion:** When we type $ in terminal and then press tab, all shell variables that are available is displayed.

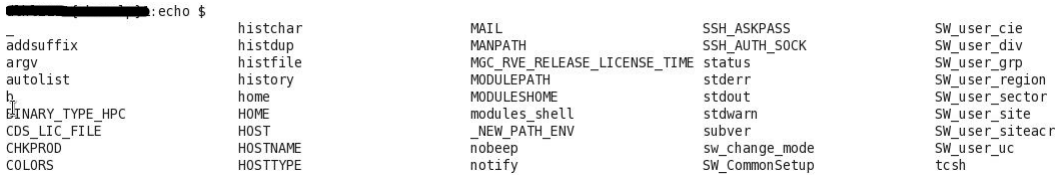

Figure 6.6: Variable Name Completion

- **Username completion:** When we type tilde (~)and then press tab, all usernames are displayed automatically.

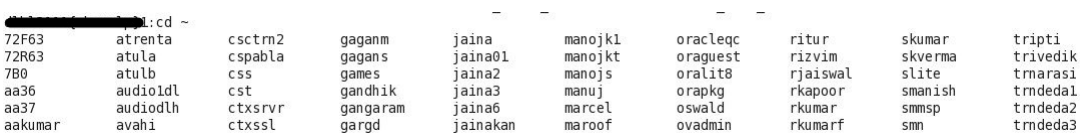

Figure 6.7: Username Completion

- **Filename and directory completion:** Display all the available files and directories.

Figure 6.8: Filename and Directory Completion

6.4.3 Proposed Completion

We have proposed a completion of name of scripts - used in monitoring process, options that are given to those scripts and values of those options.

There were two approaches available to us:

- Extract script name, options and their possible values from one file of product.
- Get script name from file name of that script and extract options and their possible values from that file.

We have implemented proposed completion in both approaches, as we have products in both of these approaches. Example of proposed completion is shown in figure 6.9

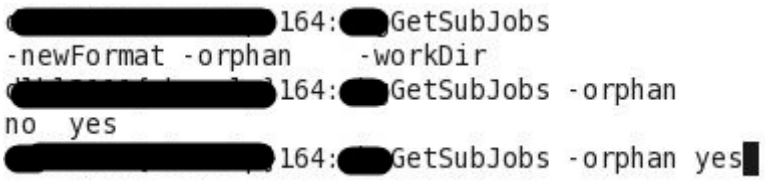

Figure 6.9: Script Option Completion

As you can see in figure 6.9, proposed completion shows possible values of given option, if values are Boolean or enum.

It also checks the type of values given to options that are provided as an argument to the script while executing.

For example if the value of option should be string and you give value of another type, it prints that "Value of 'option' should be string".

Chapter 7

Analysis

We have done monthly analysis of how many failures occurred of which type in year 2015 to 2017. The following graph shows that how many failures occurred monthly in year 2015 to 2017. See figure 7.1.

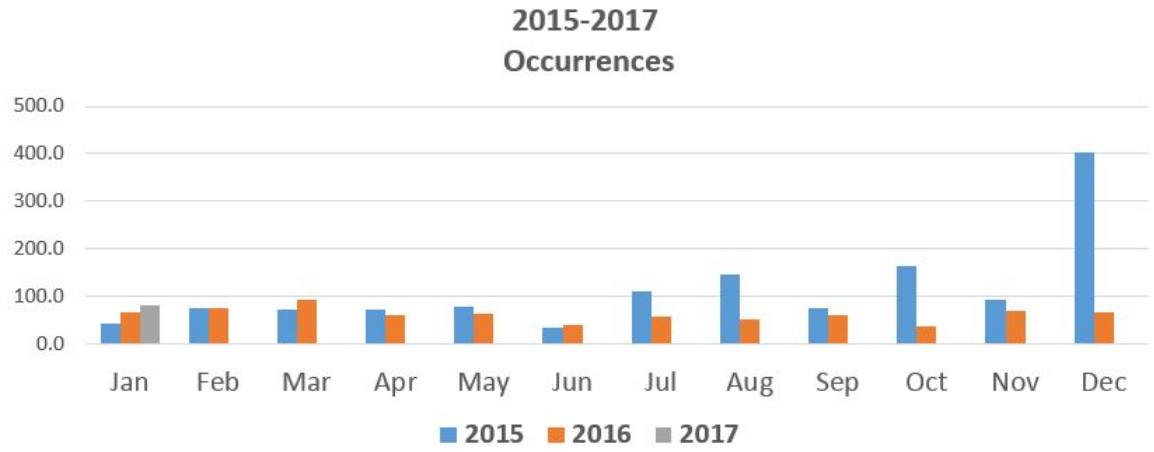

Figure 7.1: Failure Occurrences in Year 2015-2017

The below mentioned graph shows that how many failures occurred of which type in year 2015 to 2017. See figure 7.2.

The following graph shows monthly analysis of the reduction in efforts to resolve failures. See figure 7.3.

In year 2015, total 320 men days spent and out of total efforts 30% efforts spent on IT infrastructure issues. In year 2016, total 225 men days spent and out of total efforts 30% efforts reduction with respect to 2015. The following graph shows how much efforts spent to failures of which type. See figure 7.4.

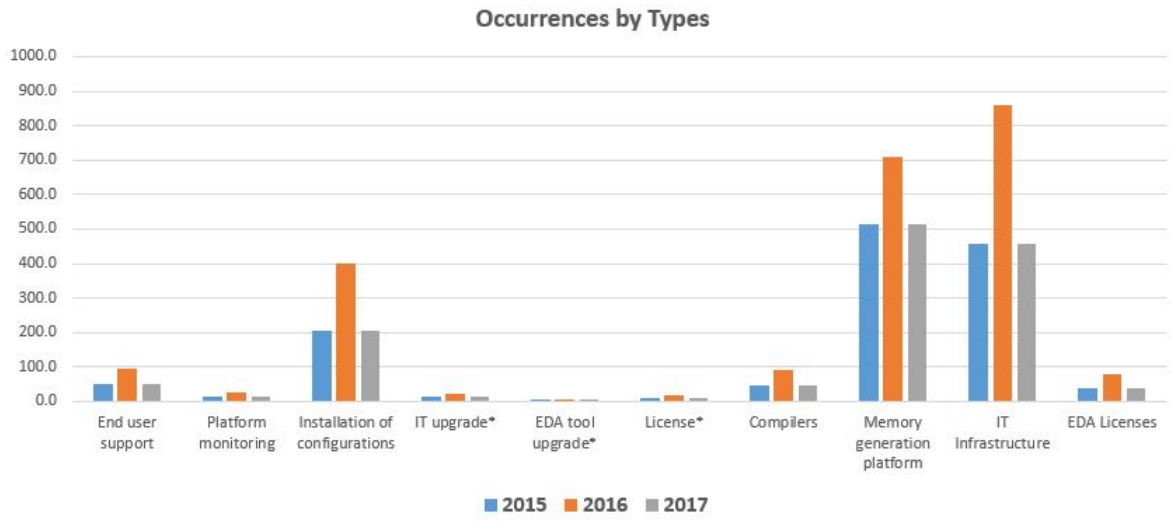

Figure 7.2: Failure of Types in Year 2015-2017

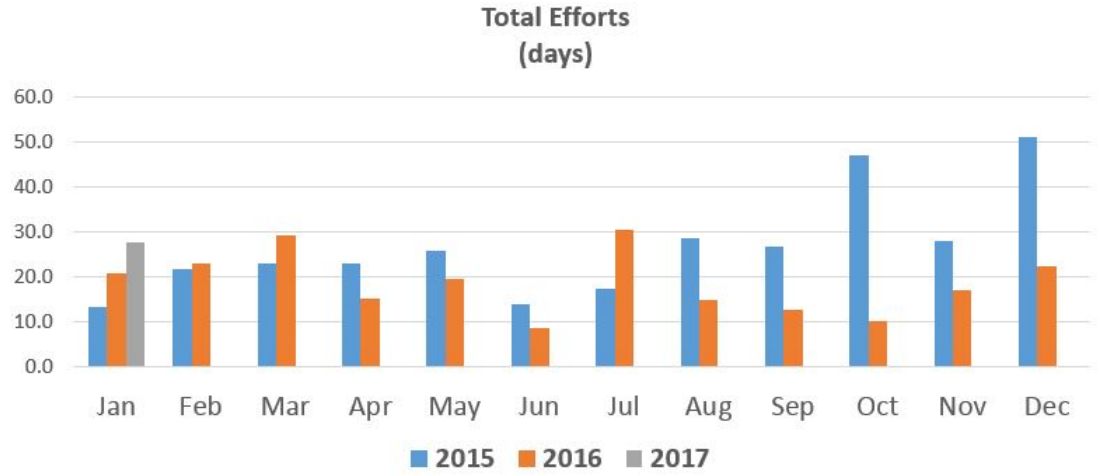

Figure 7.3: Efforts to handle Failures in Year 2015-2017

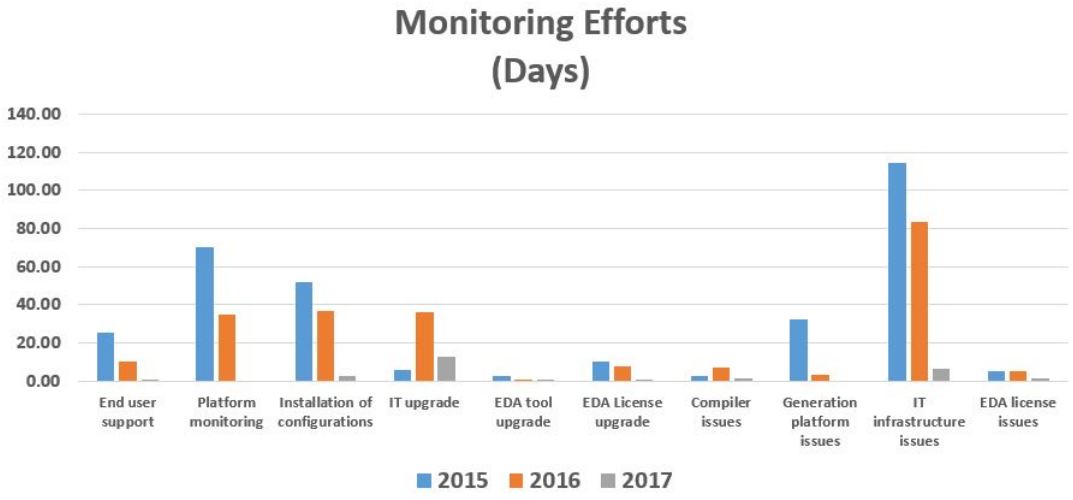

Figure 7.4: Efforts to handle Failures in terms of Types of Failures in Year 2015-2017

Chapter 8

Conclusion and Future Work

We can conclude that the work done till now complies with the goal of the project. The proposed automation has been implemented and found ideal in comparison with existing methodology. The scripts used in monitoring tasks have additionally been updated and tested with many different test cases. The scripts have been improved and optimized. Appropriate outcomes are accomplished using these scripts.

As a future work we can give GUI to monitoring process. More automation in monitoring process can be done to reduce efforts done to handle failures.

Bibliography

[1] Dave Beazley. SWIG-3.0 Documentation, 2014.

[2] IBM. Platform LSF Foundations. *IBM Knowledge Centre*, 7.0, 2009.

[3] E. Jansson and T. Johansson. Creation of standard cell libraries in sub-micron processes. *Linkopings University*, 2005.

[4] John Levine. *Flex & Bison: Text Processing Tools*. O'Reilly Media, Inc., 2009.

[5] Maciej Sobczak. C++/Tcl Documentation, 2006.

[6] STMicroelectronics. STMicroelectronics Internal Files and Training Manuals. *STMictroelectronics*, 2015.