# STATISTICS AND METRICS GENERATOR FOR WEB-BASED SOFTWARE INSTALLATION

_____

A PROJECT

presented to the faculty of

Oregon State University

_____

in partial fulfillment of
the requirements for the
degree of

Master of Science in

Computer Science

_____

by

Arvind Guruprasad

Presented March 2004

Commencement June 2004

STATISTICS AND METRICS GENERATOR
FOR WEB-BASED SOFTWARE INSTALLATION

A Masters of Science Project

by

Arvind Guruprasad

Presented in March 2004



APPROVED BY THE GRADUATE ADVISORY COMMITTEE:

_____
Dr. Saurabh Sethia, Assistant Professor,
School of Electrical Engineering and Computer Science

_____
Dr. Timothy Budd, Associate Professor,
School of Electrical Engineering and Computer Science

_____
Dr. Ron Metoyer, Assistant Professor,
School of Electrical Engineering and Computer Science

I understand that my project will become a part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my project to any reader upon request.

---

Arvind Guruprasad, Author

# ACKNOWLEDGEMENTS

I express my sincere thanks to my academic advisor and major professor, Dr. Saurabh Sethia for his tremendous support and constant encouragement he provided without which none of this would have been possible. Over the past two years, the innumerable number of technical discussions with him has helped me a lot in achieving my goals.

I express my gratitude towards Dr. Timothy Budd and Dr. Ron Metoyer for sparing their valuable time, willing to serve on my committee. I would like to thank Dr. Mike Johnson for providing me guidance throughout my graduate program.

I express my appreciation to Sun Microsystems Inc. for providing an internship opportunity during a substantial part of my graduate studies. I am grateful to my manager Mr. Bryce Schroeder, my mentor Ms. Stacey Beck and my team at Sun for their continuous support.

I wish to thank all the professors under whom I have taken courses and the administrative staff at the Computer Science department office. Last but not the least, I am indebted to my family and friends, for their love and support during my graduate studies.

# TABLE OF CONTENTS

# LIST OF FIGURES

# ABSTRACT

STATISTICS AND METRICS GENERATOR
FOR WEB-BASED SOFTWARE INSTALLATION

by

Arvind Guruprasad

Master of Science in Computer Science

School of Electrical Engineering and Computer Science

Oregon State University

March 2004

Statistics and Metrics Generator (SMG), is a software tool that gathers, stores and reports performance and execution metrics for a web-based software installation process. The purpose of this project was to develop a software utility that provides feedback about software download and installation process efficiency. A web-based software installation is prone to errors such as cache miss, incomplete download, improper installation, hardware faults etc. SMG is useful in post-monitoring the download process and providing feedback to the user. This tool parses software download information from various logs generated during installation and generates reports that provide success ratio, failure analysis and status information. SMG provides insight into how the download process executed, what errors it encountered, how much time it took for completion, and overall success percentage on a weekly, monthly and quarterly basis. It also generates charts of Operating System loads, customer specified software loads etc. The project was developed and is in production at Sun Microsystems Inc., but it is portable to any external web-based software installation process.

*CHAPTER 1*

# INTRODUCTION

**"Web-based Software Installation"**

Software can be downloaded and installed on host systems through various methods such as installation using compact discs, floppies, or by downloading from the net and extracting the setup files. All these methods involve human involvement to get the source code into the hardware system and to install it. This chapter describes the process of web-based software installation. In this approach, the software content is combined with the instruction set for installing them. These instructions are in the form of self-extracting scripts that run immediately once they are downloaded on the system. Once the rules are defined for software content that has to be downloaded and installed, it is stored in a location accessible through the internet. As in any installation, this type of installation is prone to errors such as cache miss, incomplete download, improper installation, faulty hardware etc. It is important to devise a method that tracks the installation process, and provides feedback about how it performs and executes.

## 1.1 Project Overview

This project developed software that provides statistics and metrics for the software installation process. The Statistics and Metrics Generator produces accurate and extensive performance and execution reports detailing the download process. These metrics can be used to determine efficiency and effectiveness of the process, analyze failures, resulting in swifter problem resolution and increase product quality.

The generator maintains data of appropriate software download activities that take place during installation and generates reports based on business requirements. The project is currently developed and deployed at Sun Microsystems, but can be used for other web-based software installations.

## 1.2 Project Scope and Purpose

The goal was to gather, store and report software download performance metrics. These reports are available for software loaded on various hardware products and many global locations where web-based installations are deployed. The purpose of this project is to generate reports based on business requirements and management metrics in the form of automated web generated reports, execution metrics, failure analysis, time analysis and other performance metrics. The system uses existing Sun data sources, public domain tools & standards and is consistent and portable to other web-based installations.

The SMG produces web-based charts and reports that display performance metrics such as first-pass percentage, overall yield percentage, failure numbers, volumes shipped. It also generates execution metrics such as operating systems reports, Enterprise Installation Standard (EIS) reports and system load reports. It categorizes the failure cases and types in failure analysis charting and provides root cause, corrective action and "action required" information. This tool provides feedback on the bugs in the system, indicates the need for process change and drives business decisions. All reports are converted into Post Description File (PDF) due to its universal acceptance and portability.

This chapter discusses the background that existed at Sun Microsystems, the database that stores data about the test process, the architecture of the software download process and a brief description of the rest of the report.

## 1.3 Test Process

The host computers where the software is to be installed go through a series of automated hardware tests that test their reliability and fitness for use. Once the hardware tests pass successfully, the software requested to be loaded, is installed. The installation is then tested for functionality and process correctness. Each test produces a log and is stored as a raw log in a database called the test process database.

The test process database contains data about the successes and failures of each test. It uses a relational database (Oracle(tm)) with web interfaces for viewing the data. It is possible to

gather failure or success information of the software tests as a whole. Before this project was introduced at Sun, the database was missing the sub-test level data required for providing software download metrics. The database was enriched with new fields and tables to store more specific software download specific reports

Figure 1 is the relationship diagram that explains the test process database table structure and relationship between these tables. The tables are shown inside the boxes and the lines indicate the relationship between two tables. Each table has its primary key and is linked with other tables using the foreign key. The test process database has many tables, and in the following three tables of use are discussed.

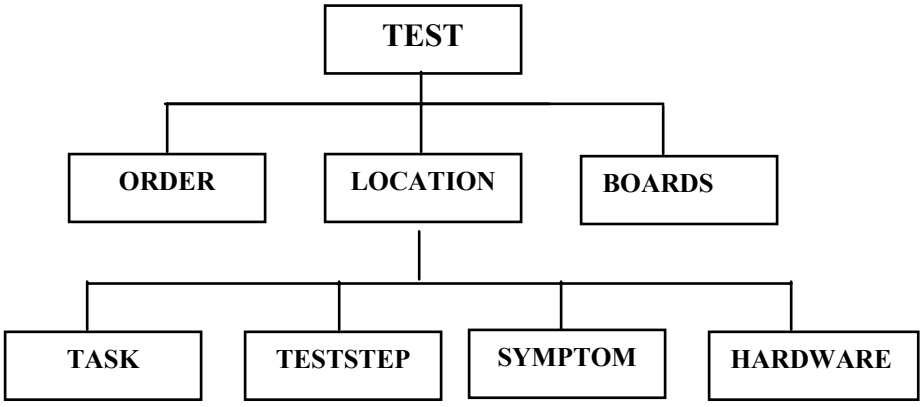


**Figure 1 – Test Process Database- Table structure**

Table Test: This table contains test number, serial number, test start time, and the operation, (what part of testing is taking place) status etc.

Table Task: This table stores information of the sub-tests. For example, software download is a sub-test inside of customer configuration tests. It contains fields such as task scripts, task status, task start time etc.

Table Teststep: This table contains information about each individual test script the sub-tests run. It contains field such as fails, passes, exit status, time to failure, symptom etc.

The database was being maintained by Sun before the project started. New fields were added

to the tables and a new table was created to store data and produce the specific reports. The new table created is discussed in Chapter Six. The architecture of the software download process is described next.

## 1.5 Software Download Architecture

In order to understand how the software download process is done, it is first necessary to have a basic understanding of the way software is typically installed. Software download in simple terms involves the following four steps.

**Download** - In this stage software is obtained from a repository server where its latest version is stored, and moved to where the installation is to occur.

**Installation** - The software is installed into the runtime environment of the host system. Once the software package is installed, it is extracted and integrated as a working component into the system.

**Configure** - The software's configuration is customized to match the operating environment as well as the customer's specific desires, for use.

**Verify** - A final check of the entire process is made to ensure that nothing went wrong or created any exception conditions.

### Download

During the download phase, software is obtained by the host computer from the software server. This software is referenced by a URL. It is retrieved by sending a request through two levels of caching. The system requests first goes to a “leaf node” cache. The leaf node cache is a moderate capacity, temporary storage cache. It is the responsibility of this cache to maintain a collection of the most recent software packages that have been downloaded so that they can be obtained again directly without having to push the request further up the chain to the "top level" caches.

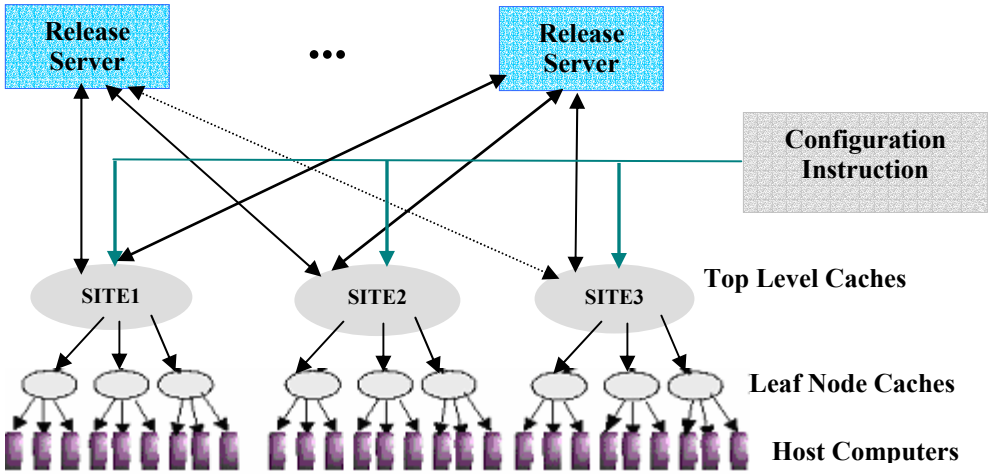


**Figure 2 - Process Architecture**

Due to the fact that each of these "leaf node" cache servers has limited disk space they will never be expected to be able to store all required and active software packages over a long period of time. For this reason, all requests that the "leaf nodes" cannot fulfill are passed to "top level". These "top level" caches are equipped with an abundant amount of storage space, adequate to store all active software packages needed at a facility any given time. For this reason, these cache servers are considered "deep" caches, meaning that all data that they receive is meant to be permanently stored until manually removed at a later time. This functionality is what gives the ability to operate within adverse network conditions. Also, each site with their top-level cache and leaf node caches can function independent of another site. This software caching system is illustrated in Figure 2. As shown in Figure 2, each of the "top level" caches receives its data from a server where the requested software is stored. The architecture provides the ability to maintain a single collection of all loadable objects, copied into the repository from remote release servers. In addition, this repository also allows control of the exact revisions of software which is the most appropriate for download, de-coupling the remote offsite servers from actual load and install processes.

### Installation, Configuration, and Verification

Once software has been received by the server on which it is to be installed, a script is then run which performs an installation of the software package. For most software bundles, this

stage consists of a simple package addition to the Solaris(tm) operating environment. This script also adds the ability to modify package parameters and verify that the software has been correctly and completely loaded onto the target system.

The software engineering project life cycle method is used to describe the rest of the project. The next chapter describes the data flow of the software download process, logs it generates, and information it contains. Then the project requirements are discussed. The following chapters deal with the high-level architecture, design implementation and testing of the individual modules comprising the project. Some of the outputs are explained and recommendations for future work are provided.

*CHAPTER 2*

# DATA ANALYSIS

**"What logs are produced, what information they contain"**

Web-based software installation involves downloading, installing and configuring software. During this process, many logs are generated at many locations. To generate the required reports, it is essential to understand the data and instruction flow in and out of the process. During the data analysis phase, the data flow of the test process and its interaction with the software download process is studied. Metric issues such as information currently available, information required, data location, data retrieval process and data storage are understood. This analysis leads to the performance and execution metrics derived from the process. Executive management was queried for what was required, and what information is important to the project.

## 2.1 Data Flow Diagram

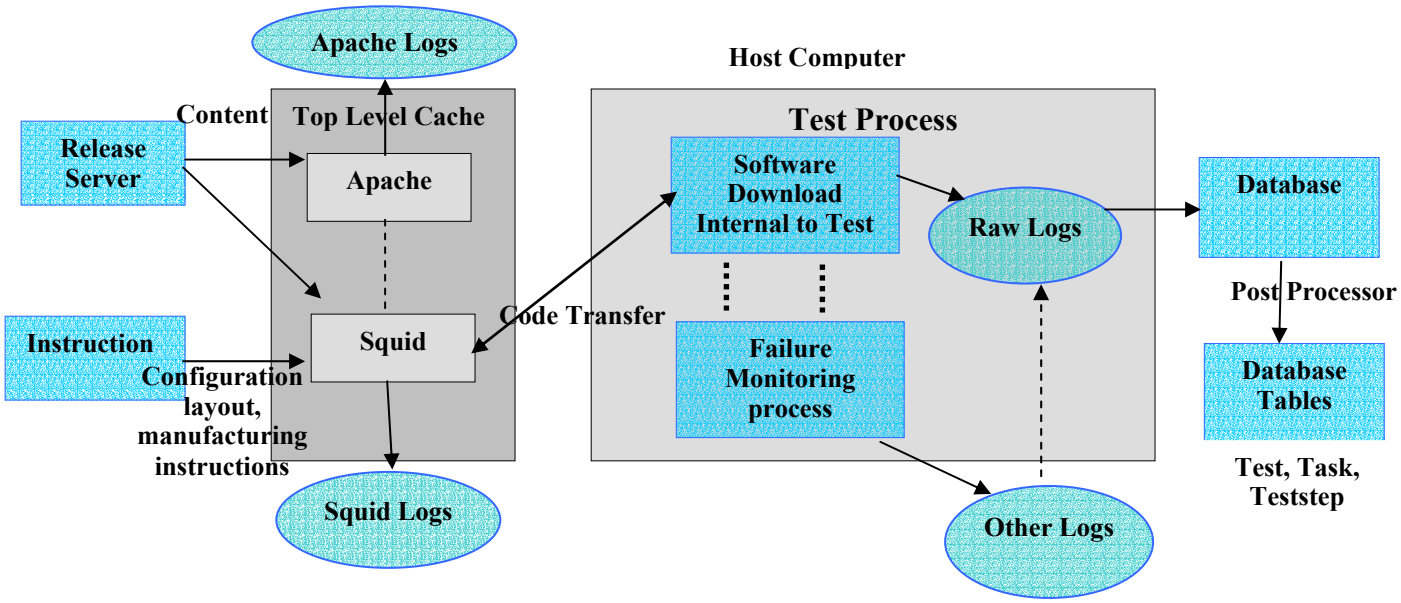


**Figure 3 - Data Flow Diagram**

Figure 3 shows the software download process as a part of the entire test process. The inputs to the test process come from the storage servers through the proxy cache. There are two types of servers, one for storing the actual software code and the other storing information about the configuration details. The test process reads the configuration information from the server, and requests the software code it requires through the web proxy. There are two types of caches, Apache and Squid. Apache cache is used for administrative details, such as identification, authentication and authorization. Squid is a free, open source full-featured web proxy cache designed to run on Unix systems. The Squid cache stores the software content. You can find more information about Squid at www.squid-cache.org. Squid generates its own logs that store information regarding the cache usage and utilization. Similarly, Apache has its own logs maintaining administration information.

Software download uses the storage server and the proxy to complete its execution. Once the entire test process is completed, a Test log (Post installation log) is generated. This contains information about how the test executed, how many errors were encountered and its status. Software download is a sub-test in the entire process and hence has its log information as a part of a longer log. These test logs are then stored in the test process database. A separate database field collector program runs through these log files and populates the requested fields in the database. In addition to these logs, logs are generated as part of Enterprise Installation Standard (EIS) install process. EIS is a standardized process of steps taken while software download is executed. Each test process has to be standardized according to a specified version of EIS. EIS also includes a standard methodology for software patch installation. The standards are updated and released on a regular basis. It is essential that the test process take place according to the latest EIS standard. The EIS logs contain information about what version of EIS was used for the test process.

The data-flow diagram provides a different view of the process and the data locations where appropriate information can be found for reporting. The following table lists the data location and the data it contains that are appropriate for reporting.

## 2.2 Data Location and Analysis

| ***Post installation logs*** | ***Data Contained*** |
|---|---|
| Customer Order Number | Customers Billing of material invoice, the system unit number tested |
| Configuration File | Configuration parameters |
| Pattern for OS version | Contains Solaris(tm) version installed |
| Domain Definition File | Domain Load details, number of domain loads, file system configured |
| Solaris(tm) Jump-start log | Process of installing Solaris(tm) |
| Enterprise Installation Standard | The defined standard of installation |

| ***Squid Cache logs*** | ***Data Contained*** |
|---|---|
| Access log | Cache Miss |
| Squid.out | Cache Downtime |
| Cache.log<br>Wget – Command to download from cache server | Download error information<br>Download time, Failure rates, percentage unavailable |

| ***Database*** | ***Fields*** |
|---|---|
| Table Test: | Test Number, Test Start time, Test End Time, Family, Operation |
| Table Task, TestStep | Task Number, Task Start and End time, Task Status, Script |
| Table BOM: | Test Number, BOM No., Part number, serial number, Description |

From these logs we have the following metrics.

**Management/Performance Metrics generated:**

| ***Metrics*** | ***Location*** |
|---|---|
| OS | Post Installation logs |
| Software Download volumes | Database |
| Percentage of success | Database |
| Number of restarts | Database, yields first pass (Number of downloads that passed at the first attempt) |
| Part numbers (unique identification number for each system on which download happens) | Configuration file |
| System serial number, Ethernet address | Post installation logs, Database |

**Execution Metrics:**

| ***Metrics*** | ***Location*** |
|---|---|
| Time to download | Database |
| Length of time to complete install | Database |
| Download Mbps | Cache logs |
| Percentage unavailable | Cache logs |
| Failure rates | Cache logs |
| Cache Misses, Cache downtime, Miscellaneous Caching systems | Squid access, cache logs, squid.out |

*CHAPTER 3*

# PROJECT REQUIREMENTS

**"What do we need to do"**

The requirements gathering set the tone and direction for the development of the project. Identifying the problem domain is imperative to get an efficient solution. The primary requirements were the management and execution metrics. The customers who use the tool and their requirements were recognized.

The following customers were identified.

1. Executive Management – They comprise group of Vice Presidents who are responsible for taking business decisions based on performance and execution.
2. Software Integration Engineering Team – This team is responsible for the actual web-based download process. Feedback is required on error numbers, error types, success percentage, drives process changes.
3. Manufacturing floor technicians – They are responsible for any manual intervention to the test process. When a test fails, they view the failure analysis viewer to get the failure information, make necessary modifications to the test process and retest them.
4. External software downloads teams –The software download process is in production at Sun. In the future, when the process is deployed externally, the report generation will be ported and available externally as well.

**Project Specification:**

1. Management metrics such as success rate, overall yield, failure numbers, volumes downloaded and failure analysis. These reports are used to get information of how the architecture works, how efficient it is, and drives scope for improvement.
2. Execution metrics such as operating systems loaded, the install standard version loaded, number of domains loaded etc. These reports provide more information for management and decision support. For example, we can know which operating system

is popular etc.

3. Automated report generation that updates itself daily. The reporting information should be valid and updated on a daily basis without any manual intervention.
4. Report Presentation – One important requirement is that the reports should be web generated. Though there are many ways in which the reports can be presented, PDF format was selected, since it is universally accepted and is platform independent.
5. The metrics system should be portable to other web-based installation system at other external locations.

**Resources required:**

Determining the software and hardware resources for implementation

Software- Since it involves data parsing from log files, storage of reporting information and reporting front-end display multiple software languages and utilities were chosen for these multiple purposes.

- Perl – for parsing and data collection. An existing parser was used, and necessary modifications were made to suit the needs.
- Oracle(tm) - SQL database for storage. The existing database structure was modified by adding fields and tables.
- Java(tm) - for report implementation. A Java(tm) based open source existing reporting tool called Jfree, a free Java(tm) chart library was selected. JFreeChart supports pie charts (2D and 3D), bar charts (horizontal and vertical, regular and stacked), line charts, scatter plots, time series charts, high-low-open-close charts, candlestick plots, Gantt charts, combined plots, thermometers, dials and more. JFreeChart can be used in applications, applets, servlets and Java(tm) Server Pages (JSP).
- Software tools such as Ant for building, Concurrent Version Systems (CVS) for revision control.

Hardware – Existing resources that were being backed up were used that includes an

internal server in Hillsboro-Oregon, a database server and a web server. An internal server was used to run the Unix utility "Cron" each day to run the code, collect the data and generate reports automatically. The reports were stored on web servers on a long term basis.

*CHAPTER 4*

# HIGH LEVEL ARCHITECTURE

### "What Modules"

The design can be broadly classified into three sections as shown in Figure 4.

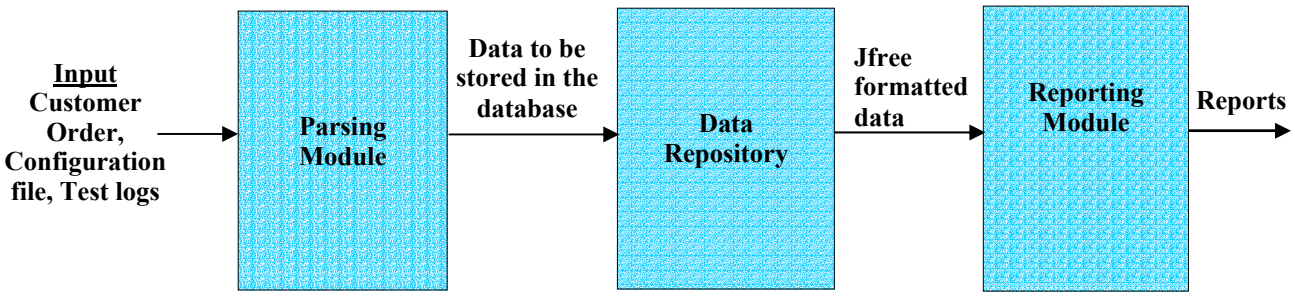


**Figure 4 - Design Block Diagram**

The input to the software download process comes from the customer order that specifies the combination of software selected, and the configuration details of how the software needs to be loaded. Once software download is performed, logs of how the process went are created. These test logs are analyzed by the parsers for consistent patterns, and regular expressions matching these patterns are created. This grabs the data fields and stores them into the Oracle(tm) database. The data capture is done in parallel with the running of the test process, or as a post-process. There are two types of parsers, the failure analysis parser and the post-parser. The functions of these parsers are described in the next chapter.

The second module is the data repository. Oracle(tm) test process database was used for storage. A new software download table was created within this database which uses the existing features and adds new fields and relationships for future reporting. Data files pertinent to reporting were also created and stored in a CVS controlled repository. The use of these files will be explained in detail in chapter 6.

The third part is the reporting module. A Java(tm) based reporting solution that processes the

data, constructs reports in the format required and generates automatic reports. This module also involves a website for posting and displaying the reports.

The following diagram represents the detailed design architecture:

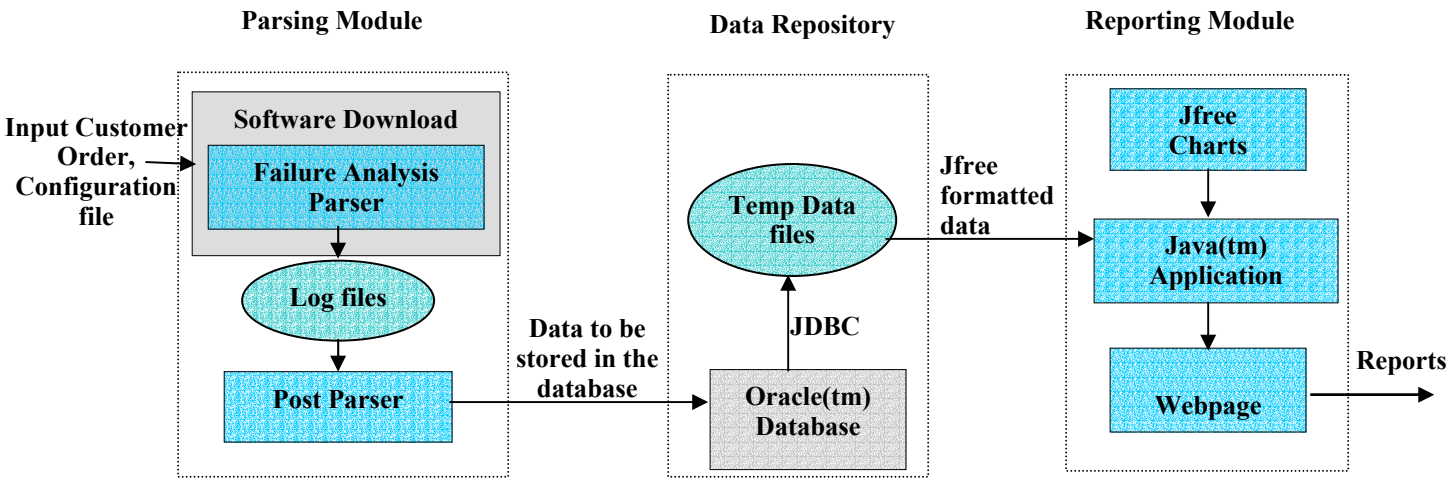


**Figure 5 - Detailed Design Diagram**

Figure 5 describes the detailed design diagram. The dotted lines mark the three modules presented in the high level design. The test process involves hardware and software tests. During the software tests, a parser concurrently scans through the log files that were created. If it finds a failure pattern, it grabs the pattern, and stores it in a table, and stops the test. Using this failure table, it categorizes the errors into the various types. After the execution of the software tests, the complete log files are produced. The post-parser also scans the log for other information necessary for reporting. All the information parsed is stored in the database in the form of tables and fields. The third module is the actual report generator which uses the data, converts them into a specified input format and develops the reports. A special Java(tm) based reporting tool called Jfree was used. (Jfree is explained in chapter 7)

In the next three chapters, these three modules are described in detail. They describe the individual design internal to the modules, their implementations and their testing details.

*CHAPTER 5*

# PARSING MODULE

**"How are the logs parsed?"**

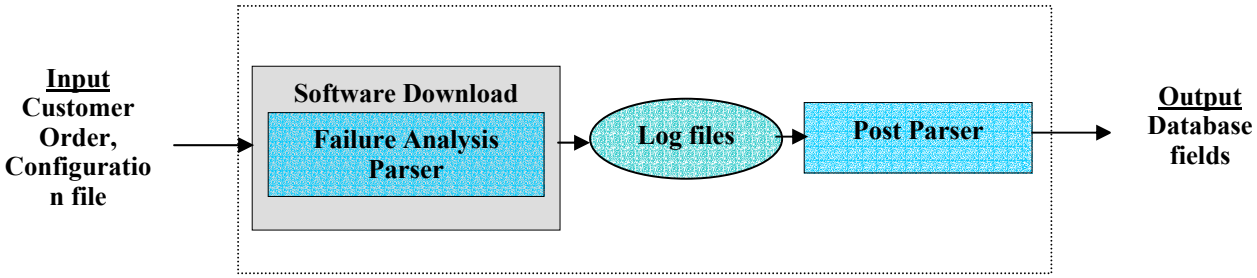


**Figure 6- Parsing data flow**

This chapter describes in detail the design, implementation and testing of the parsers that grab essential data and store them into the database. Figure 6 explains the parsing data flow. These parsers take their input from logs generated during and after the test process, parse the logs for required information and store them as database fields. There are two types of parsers, the failure analysis parser and the post parser.

## 5.1 Failure Analysis Parser

The purpose of this parser is to run in parallel to the test process, and collect failure information during the software download process. This also provides feedback to the download process about the failure and halts it if necessary. The statistics and metrics generator generates data on the number of systems failed during installation. But the information of how the system failed was not available. In order to develop a chart of the failures, the failure analysis parser was developed. It has a mapping table set up with all the possible failure patterns and the corresponding categories they fall into. The failures that occurred during the last six months were collected manually by searching through logs, analyzed and a broad classification was developed. In this particular installation, we have failures categorized into Infrastructure, Hardware, Software, Process and Unknown. The failures of the last six months were classified into one of these categories. The mapping table

contains these failures and maps them to one of these failure categories. For each of the failures, a regular expression was created. The parser is a script that waits for a match of any of these failure patterns from the log files. If a match occurs, it maps it with the help of the mapping table and correlates it with the corresponding failure category. The mapping table also contains a corrective action for each of the failures recognized. This script also keeps looking for patterns not readily available in the mapping table. In such a case, it grabs the failure information from the logs, and informs the technician about this newly matched failure. The technician or the administrator then finds the root cause, and corrective action needed. This new combination of failure, root cause, corrective action was then added to the mapping table. The failure was also categorized into any one of the existing buckets.

A name value pair was then created with information such as Root cause, Corrective action required, and the action taken. This name value pair was appended to the end of the log files so the post parser can grab them for permanent storage into the database.

### 5.2 Post Parser

The output of the test process which involves software download is in the form of log files. These logs are then stored in a database. These raw logs do not provide data in a convenient form. Hence, a mechanism to grab the information we need from these logs and store them in the form of database fields was required. For this purpose, the post parser is developed. This parser runs a query to select the group of log files we want to search, opens each log file, checks for a series of regular expression patterns to spot the name NAME=VALUE pairs, and stores the value in the corresponding database field. It acts as an interface between the log files and the database. It also spots inconsistencies in the expected values from the log files and informs the administration. This feature helps in identifying irregular patterns from log files, capture them so that the administrator can create a new match pattern. Perl is selected for this parser because it is one of the best text processing languages available. It also has a strong database linking feature, the Perl-DBI. Now the post parser algorithm is described.

### 5.3 Post-Parser Algorithm

1. Set up all environmental variables required. ORACLE(TM) HOME, PATH and ADMIN variables.
2. DBI- connect to the Oracle(tm) database for specified user, password.
3. Create SQL query to extract required log files, execute query.
4. For each log file returned do Step 5 to 9.
5. Open the log files, print error if not able to open.
6. Search for patterns in the form of regular expressions for each of the database field to be populated.
7. Grab corresponding matches, store them in variable names.
8. If irregular pattern match occurs, update regular expressions with new information.
9. Close the log file.
10. Update database fields with corresponding variable names.
11. Disconnect from the database.
12. Send output, error messages to admin email.

The testing revolves primarily around the different regular expressions for different failures. There might be cases in which the regular expressions do not grab the required data, or there might be an instance of more than one regular expression returning a value. Certain test cases also involve the database connection while updating the database. Here are some test cases.

### 5.4 Post Parser Test cases

The test cases that counter the following scenarios were written.

1. The database server is down.
2. Database is not the expected one.
3. Opening non existent or corrupted log files.
4. Regular expression does not match expected value.
5. Regular expression matches, but returns a string not existing in the mapping table.
6. Query returns null value or is unable to execute.

7. Updating database without admin rights.
8. Database rollback and commit.
9. Setting alert for unknown error messages.

*CHAPTER 6*

# DATA REPOSITORY MODULE

**"Where is the data stored?"**

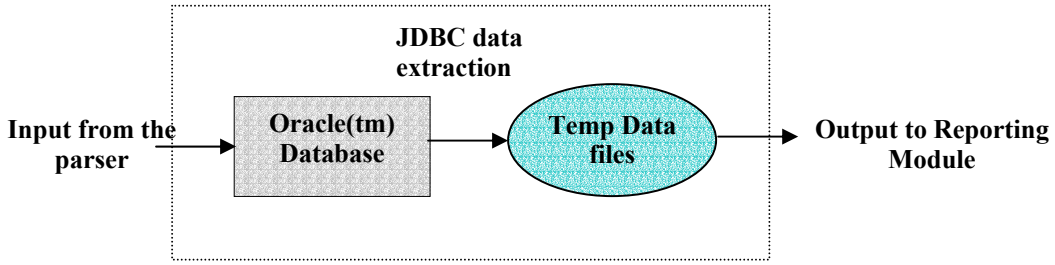


**Figure 7: Data Repository module**

Figure 7 refers to the data repository module. This module involves storing data from the post parser, extracting information from the database using JDBC, and providing them as input to the report generator. Before it gets to the report generator, the data has to be converted into a format required by JFree. The data is extracted from the database and stored as temporary data files containing name value pairs. These data files are then fed to the reporting module.

Since the test process database did not initially contain all the information needed, a table with new set of fields was created.

## 6.1 Database design modifications

To the test process database, a new table with fields that store additional sub-test level information for the software download process was added. Figure 8 shows the extra table added. It shows the initial database tables in black and the new table added in blue. It also shows the fields of data, the new table stores.

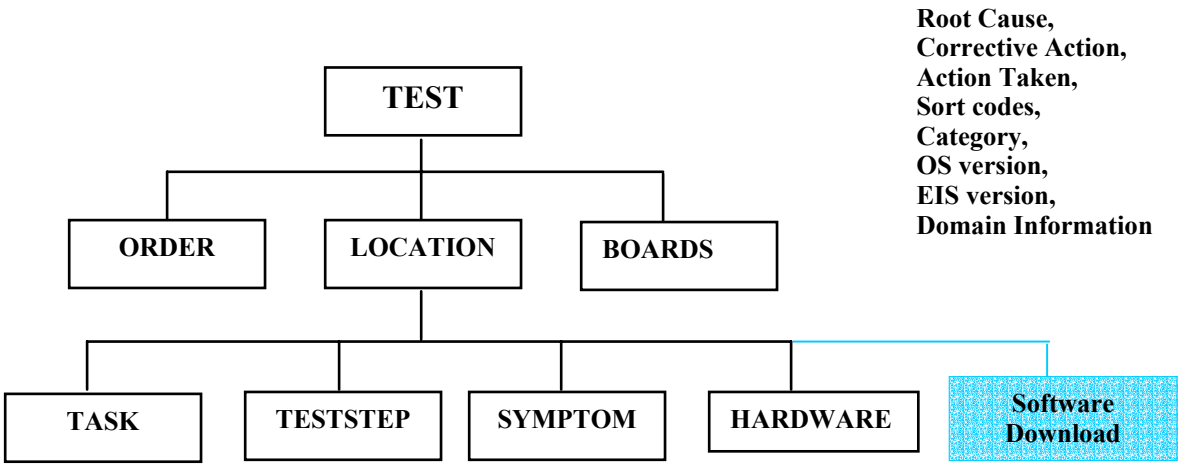


**Figure 8 - New table added to the table design of database**

Figure 8 shows the new table structure with the new table added and the fields they contain. The fields root cause, corrective action, and the action taken was used for generating failure charts. The sort code is a handy field used for sorting the failures on various criterion. The category field denotes the failure categories. This table also stores information such as the Operating system loaded, the Enterprise Installation standard (EIS) loaded and the various domains loaded with individual specification. It also provides a link between the output of the download and the initial customer order requested. This provides a feedback on what was asked and what was executed. Oracle(tm) is used as the database. Since most of the database fields required existed, the existing resources were used and new specification was added where required.

## 6.2 Temporary data files

There are three different types of temporary data files. They are data files which contain data elements extracted from the database through Java(tm) Database Connectivity (JDBC), property files that contain configuration details, and the query files that contain the actual queries used for extraction.

*A data file example*

The data files stores information from the database in the format needed by JFree. The following is the format of the data files.

Data.1.Name = Data.1.Value
Data.2.Name = Data.2.Value
Data.3.Name = Data.3.Value
Data.4.Name = Data.4.Value

*A property file example (configuration details)*

The property files contain configuration details such as site information, product information, database connection information etc. It looks like the following.

Site.1.Name = Site1
Site.1.Location = Location 1
Site.1.Connection = db.connection
Site.1. Username = Username
Site.1.Password = Password
Site.1.Inputfile.1= Path to input file 1
Sie.1.Outputfile.1 = Path to output file 1
Site.1.Inputfile.2= Path to input file 2
Sie.1.Outputfile.2 = Path to output file 2
Site.2.Name = Site2.

The reporting module has code for opening up a database and for opening and closing input and output files. The connection and data files were then specified by the configuration file. The advantage of this method is abstraction of the code from the configuration details. For example, if the IP address of the database changes the property file can be modified to address the change. This avoids code modification.

*Query files example*

The queries files are also created as property files. The query file format is as follows.

Total_tested = select count(unique(serialno)) from test where teststarttime > startdate and teststarttime ≤ enddate and family = product1 and operation = software download

Total_pass = select count(unique(serialno)) from test where teststarttime > startdate and teststarttime ≤ enddate and family = product1 and operation = software download and teststatus =’P’

Total_fail = select count(unique(serialno)) from test where teststarttime > startdate and teststarttime ≤ enddate and family = product1 and operation = software download and teststatus = ’F’ or teststatus = ‘A’

The reporting module opens these query files and iterates through the “Name = Value” pairs, thereby executing each query. Again, the queries are abstracted from the code. So any modifications external to the code such as database design changes, query changes etc do not affect the code. The next chapter discusses the third module, the report generator.

*CHAPTER 7*

# REPORTING MODULE

**"How do we generate reports?"**

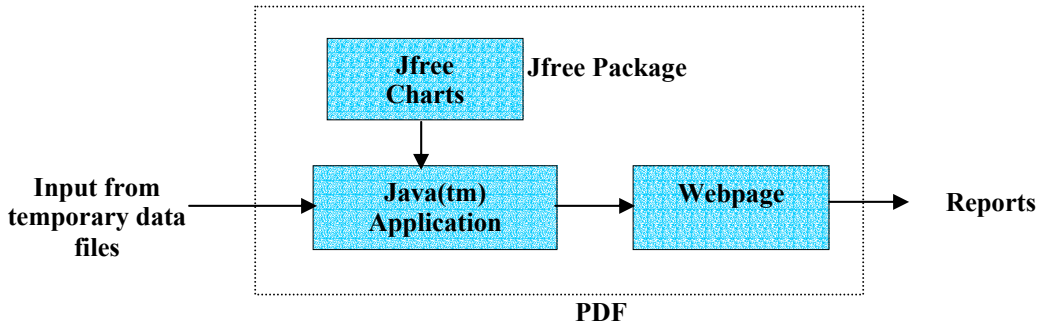


**Figure 9: Report Generator Design Diagram**

The purpose of this module is to generate statistics and metrics on the software download process. As shown in Figure 9, the application interfaces with the database, producing charts from the queried data and presenting them in a web-based format. The selection of Java(tm) was natural since it provides an excellent connectivity to Oracle(tm) through Java(tm) Database Connectivity (JDBC), and has plenty of open-source reporting tools available. For reporting and chart creation Jfree, a free open-source Java(tm) based reporting tool, was used. This package provides options to create charts in the form of applets, servlets and PDF files. The disadvantage with producing applet based reporting was that they are browser dependent and have security issues. A Servlet based solution requires a server running all the time. PDF provides a handy way of generating reports that are browser independent, and do not need a server running. PDF is supported across all operating systems and browsers. Jfree provides a package that creates new PDF files, imports the charts into them and saves them in a web location. So the report generator runs the queries, collects data , uses Jfree to generate charts, converts them into PDF files and stores them in a web location. Then a web-page for looking at the charts was designed using Twiki, HTML and Java(tm) script.

### 7.1 Jfree – Java(tm) based reporting tool

JfreeChart is a free Java(tm) chart library. JFreeChart supports pie charts (2D and 3D), bar charts (horizontal and vertical, regular and stacked), line charts, scatter plots, time series charts, high-low-open-close charts, candlestick plots, Gantt charts, combined plots, thermometers, dials and more. JFreeChart can be used in applications, applets, servlets and JSP and can create PDF files with the charts embedded. It provides complete source code, under the terms of the GNU Lesser General Public License. It provides access to data from any source via dataset interfaces, support for multiple secondary axes and datasets and tooltips, zooming, printing. It also facilitates direct export to PNG and JPEG, export to PDF via iText , support for servlets, JSP, applets or client applications. The documentation is available from the comprehensive Javadocs. Since Jfree source is available, it was possible to override certain reporting features, add new color schema, and create new non-existent reporting features. Jfree required the data files to be a specific input format. So the data collected from querying the database, was converted to Jfree specific temporary data files and was given as input to the Java(tm) application.

### 7.2 Object Oriented Java application

The Java(tm) application has two main modules. The Data Generator, and the Report Generator. The Data Generator involves querying the database, collecting required data, converting them into temporary data files for Jfree purposes, and providing input to the Report Generator. This part needs to interface with multiple databases at each download site location. It also needs to query information about each product at each location. The Report Generator takes in the data in the form of Jfree specified temporary data files, uses the reporting packages, creates the required charts, enhances them as necessary, imports them into PDF format and stores them in a web location. The design of each of these modules is explained in the upcoming sections.

## 7.3 Data Generator

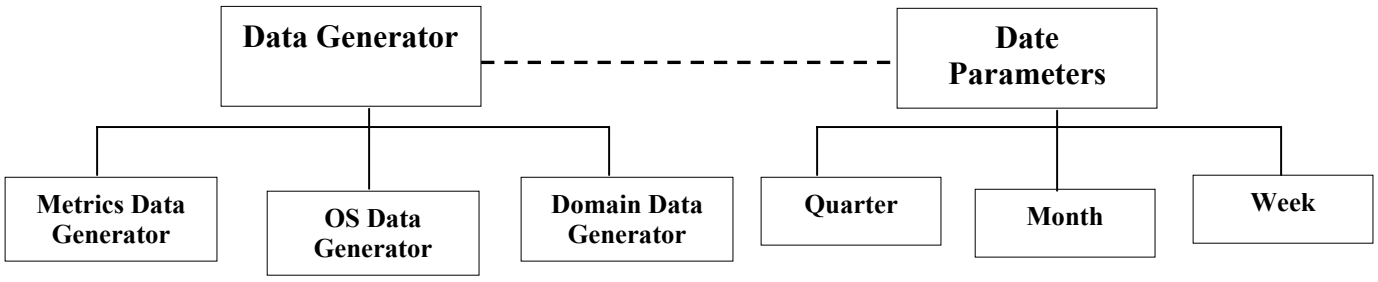


**Figure 10 - Data Generator Design**

Figure 10 describes the Data Generator class diagram. Data Generator is developed as the parent class. It contains the JDBC querying mechanism, database connectivity information, and conversion into Jfree specified temporary data files. Metrics Data Generator, OS Data Generator and Domain Data Generator are classes that extend the base class. These classes specify their respective queries through configuration files, extract information from the database and process the data into required information. Date Parameters is an interface created to provide date functions. Since the need was to create charts based on quarter, month and week, this interface provides the start date , end date and the period of time. It also derives the fiscal year of the date. Quarter, Month and Week Date Parameters are classes that implement the base interface.

## 7.4 Data Generator Algorithm

1. Set up all local variables, file streams, input readers.
2. Define collection list, with each list representing a date parameter. (quarter parameter, month parameter, week parameter)
3. For each item in collection list do step 4 to 15.
4. Get start date, end date, time frame, extension type, and number according to the parameter.
5. Open property files defining the input query file, site location and database

connectivity details.

6. For each site in the property file do step 7 to 14.
7. Set up strings Name, Connection, Username, Password, ClassDriver.
8. Connect to the site's database, set up driver manager.
9. For each product for that site do step 10 to 13.
10. Open the query property file for that product, site.
11. Create PreparedStatement with the query.
12. Execute the query.
13. Store all the values from the result set, open the output file, store output as name value pairs, Query=Result pairs.
14. Close database connection.
15. Convert data in output files into Jfree specified format, name the file unique for each time frame, product, site, year combination.
16. Close property files, exit the program.

## 7.5 Report Generator

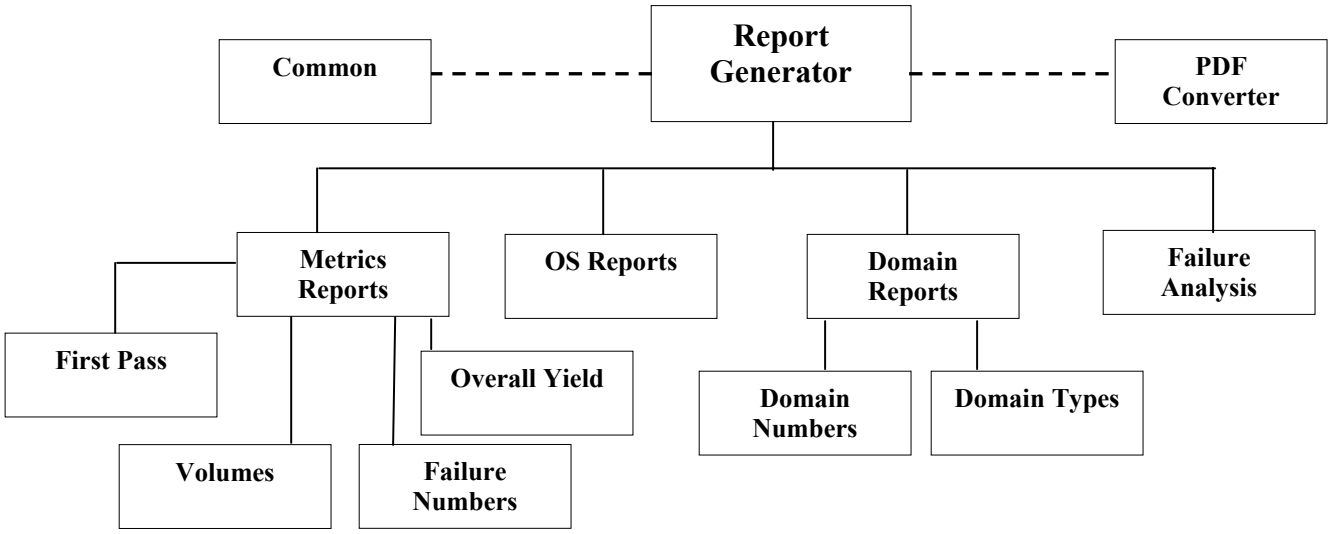


**Figure 11 - Report Generator Design**

Figure 11 describes the Report Generator class design. Report Generator was developed as the parent class. It runs through the variety of data files, as per product, site, and the time frame involved. It specifies the input data files through configuration files, and the type of report needed to be created. Metric Reports, OS reports, Domain Reports and Failure Analysis reports extend this base class. Each of these classes read their respective data files, creates chart types, runs Jfree reporting, and creates the chart. Metric Reports in turn have four classes extending them. First Pass charts – These provide information of how many units passed the test the first time. Overall Yield charts are those which give the overall performance yield which is derived from the total units passing all the tests over the total units tested. Volumes denote the volumes of products shipped by the respective site location, and failure numbers provides numeric information of how many failures happened. OS reports generate pie charts and bar charts of the OS type loaded on to the systems. A host system can be partitioned into several sub-systems independent of each other. Each sub-system is called a Domain. Each domain can be configured with different operating systems, software etc. Domain Charts report the percentage of customers wanting domain loads and the number of domains they request. Failure Analysis charts report the types of failures that have happened. All these charts provide information in various time frames such as total, quarterly based, monthly based and weekly based.

A generic class called common class contains some code that was reused by all the classes. The PDF converter uses iText.jar, a package that helps creating PDF files and imports the charts into PDF files. It also provides a handy text page having a demo chart and details of how the chart can be interpreted.

Object Oriented Programming has been extensively utilized. The design involves Inheritance (classes inheriting qualities from parent classes) to provide code-reuse and reduce redundancy, Polymorphism in the form of operator overloading, and function overloading (for example, same code used for various time frames, by changing parameters involved, same function code using different parameters and same operators performing different operations). The data is abstracted from the functions by securing the data generator away from the reporting. The special feature of “Property files” in Java(tm) was extensively used. This helps

in having the same code configured for multiple input conditions.

### 7.6 Report Generator Algorithm

1. Set up all local variables, File streams, input readers.
2. Define collection list, with each list representing a report type (First Pass, Overall Yield, Failures Numbers, Volumes, Failure Types, OS loads, Domain Reports).
3. For each item in collection list do step 4 to 11.
4. Set up report type variable depending upon the parameter in the collection.
5. Open the property file specifying all site, product input file locations.
6. For each product in property file, do step 7 to 11.
7. For each site in property file, do step 8 to 11.
8. Get data for that site, product combination.
9. Generate charts with the data.
10. Name the chart file, add enhancements, customize the chart.
11. Call PDF converter to create a PDF file, import the chart and save the chart in the current directory.
12. Close property files, exit the program.

### 7.7 Tools and Scripts

Ant, a Java(tm) based build tool, was used to execute the set of Java(tm) files. Each Java(tm) file needs its own set of ".jar" files to execute. A XML based solution was required to implement this. A "build.xml" utility was created that defined all the files and their hierarchical dependencies. Ant is a utility similar to the make file. All source files were stored in "src" directory from the main directory. All configuration files, query files, support .jar files were stored in "lib" directory. When Ant is run, it looks at the build.xml, compiles all the Java(tm) source files, links them to corresponding configuration files from lib directory, and executes the programs.

An important criterion was that the reports have to be updated each day. Since it's a live up to date report generator, it needs a mechanism through which it runs daily at a specified time.

For this purpose, the UNIX utility “Cron” was used. Cron is a command, which runs an executable from a specified location on a specified time of day, or week, or month or year. This Cron is run at midnight each day. So during that time, the Cron initiates the Ant tool. The Ant tool, builds the code with all the dependencies, and creates the charts in PDF format. The Cron then moves these PDF reports to a web-location from where they can be accessed at all times.

The code, the temporary data files, configuration files, property files, and queries were stored under a CVS repository. The Ant tool updates this CVS repository each time before running. So whenever a change was made to a tool, the Ant re-compiles them and executes the latest code. All changes and modifications were stored under this repository under various version names for tracking collaborative team work.

### 7.8 Report Generator Test cases

Test cases were created to test the module behavior for the following expected outcomes.

1. Opening input property files, query files, FileNotFound exception, File cannot open or IOException.
2. Setting up Database connection .
3. Query returns null value or is unable to execute, Resultset returning null values.
4. Returned data values are of expected type.
5. Result set domain and range limits.
6. Unexpected return values and types from function call.
7. String variables initialization.
8. Null pointer exceptions in Arrays.
9. Check if data is already available in existing arrays. If yes, add to existing values, else store the new value.
10. Division by zero error while denominator of percentage equation is zero.
11. Calculation of percentage, and testing percentage range between 0 and 100.
12. Date Testing, given a date, generation of quarter, month, week number, start date of

time frame, end date of time frame. Testing for unusual date occurrences such as end of year, leap year, week boundaries, month boundaries and quarter boundaries.

13. PDF creation with Null Charts, with no default font mapper.
14. Chart output dependent on time frame. (week, month, quarter, year)

Numerous amount of bugs were trapped by these test cases. Most of them involved exception handling, unique cases with number arithmetic and date arithmetic, graphic rendering of the PDF files, skewed data output from queries, and permissions to open reporting files. Regression testing was done manually and executed using the Ant build scripts. The next chapter discusses the outputs from the report generator

*CHAPTER 8*

# OUTPUTS

**"What reports are generated?"**

## 8.1 Overview

The outputs are in the form of reports in PDF format. They are available via web based delivery. Once the application is run, the PDF's are transferred to a web server from where it is accessed. The various reports include Metrics reports such as first pass percentage, overall yield, failure numbers, volumes shipped, OS version loaded, domain numbers and types loaded, failure analysis. They are displayed in the form of pie charts, 3-D pie charts, bar charts, line curve mapped on bar charts, and pareto charts with cumulative curves. Here are some snapshots of the charts along with their descriptions.

## 8.2 Snapshots

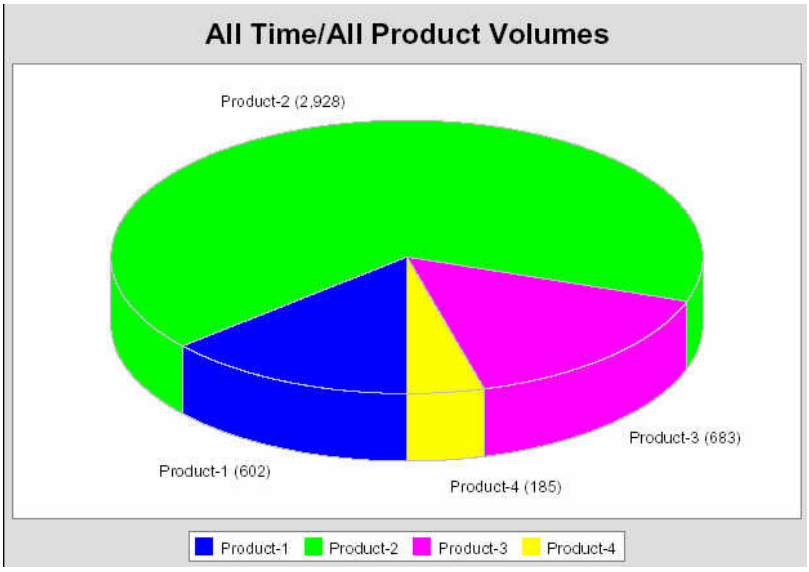


**Figure 12 - All Time/ All Site/ All Product Volumes**

Figure 12: shows the volumes of different products produced up to date from the beginning of the fiscal year across all sites.

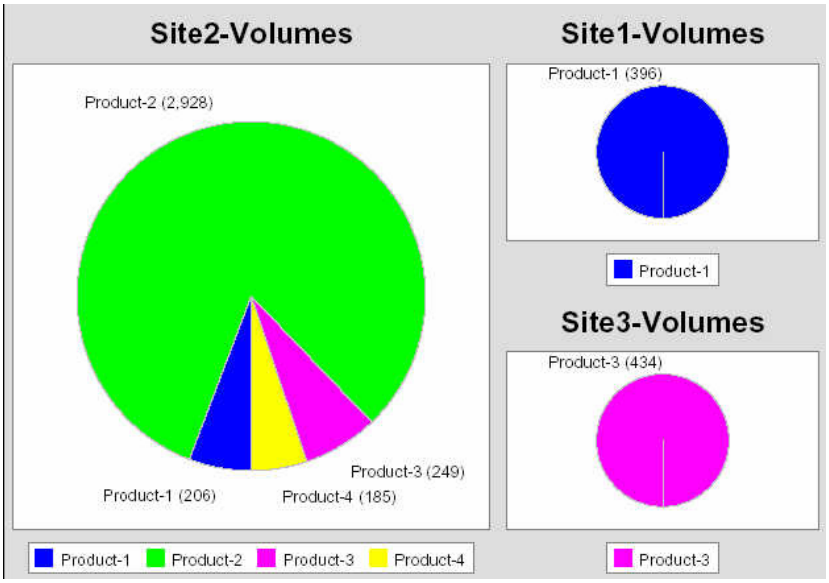


**Figure 13 - Listing of all time volumes by each site**

Figure 13: lists the volumes of each product produced at each site.

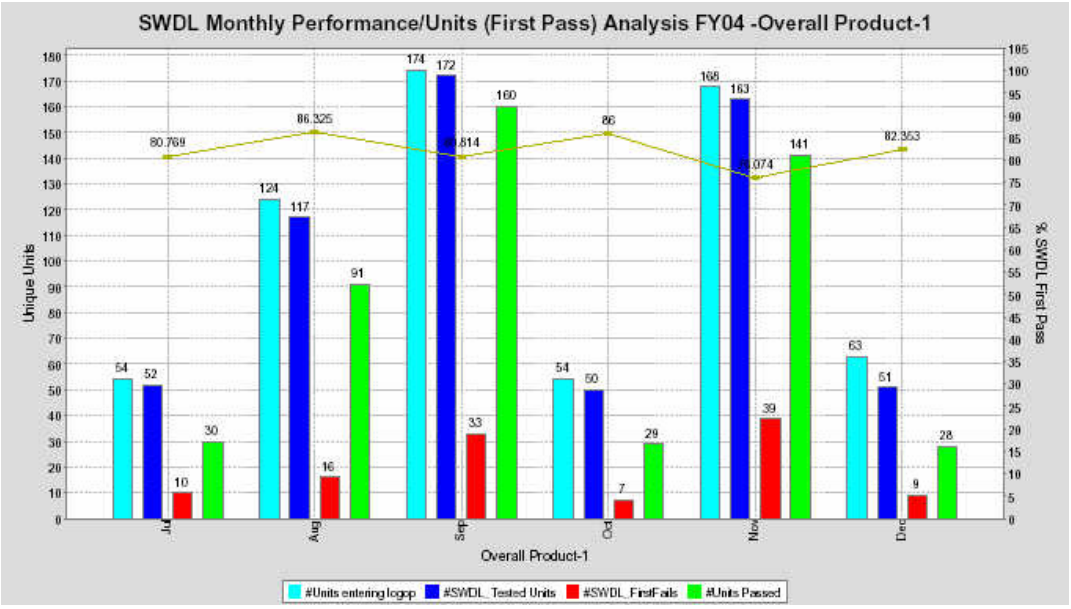


**Figure 14 - First Pass Efficiency chart**

Figure 14: bar chart depicts the unit systems entering test process, unit systems entering software download, units failing at the first attempt, and overall units passing all tests

involving software download. The line curve embedded on top depicts the First Pass Percentage.

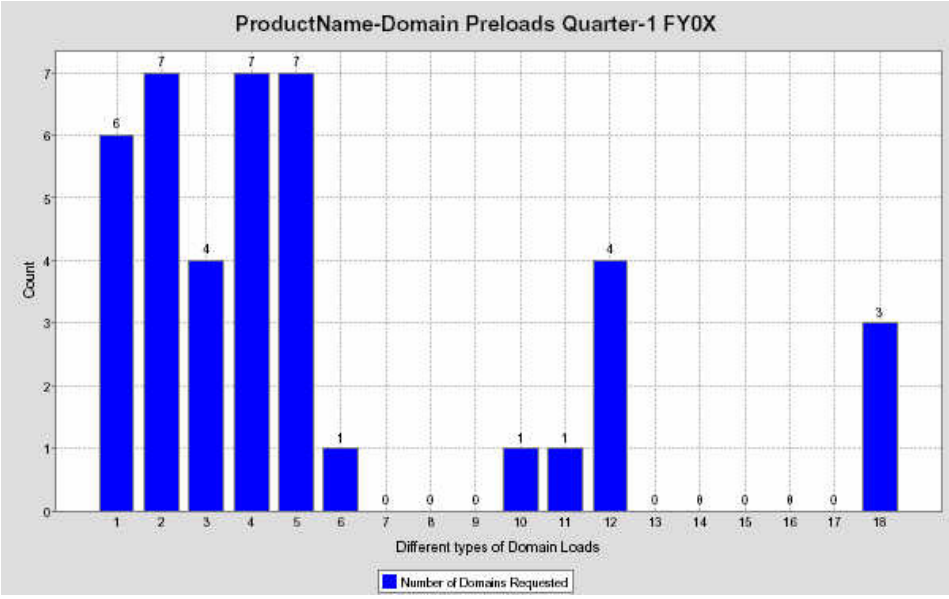


**Figure 15 - Types of Domain Requested**

Figure 15 depicts the number and types of domains requested by the customers. A customer can request between 1-18 extra domains other than default system load. This chart shows how many customers requested whatever types of different loads.

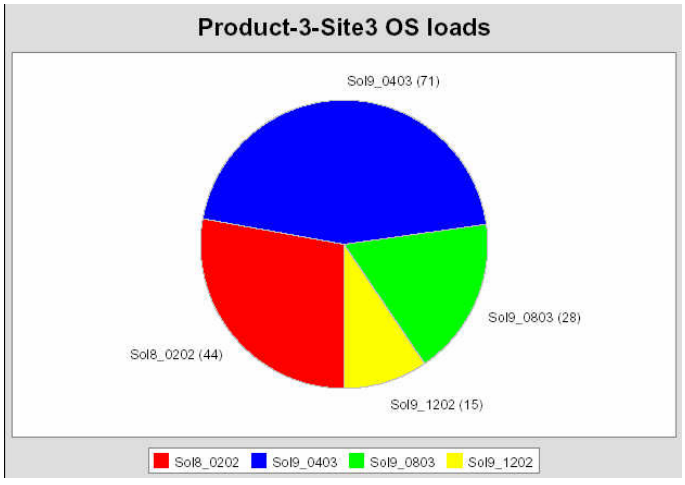


**Figure 16 - OS system loaded**

Figure 16 shows the fraction of systems loaded with different OS versions. The labels denote Solaris(tm) version and release dates.

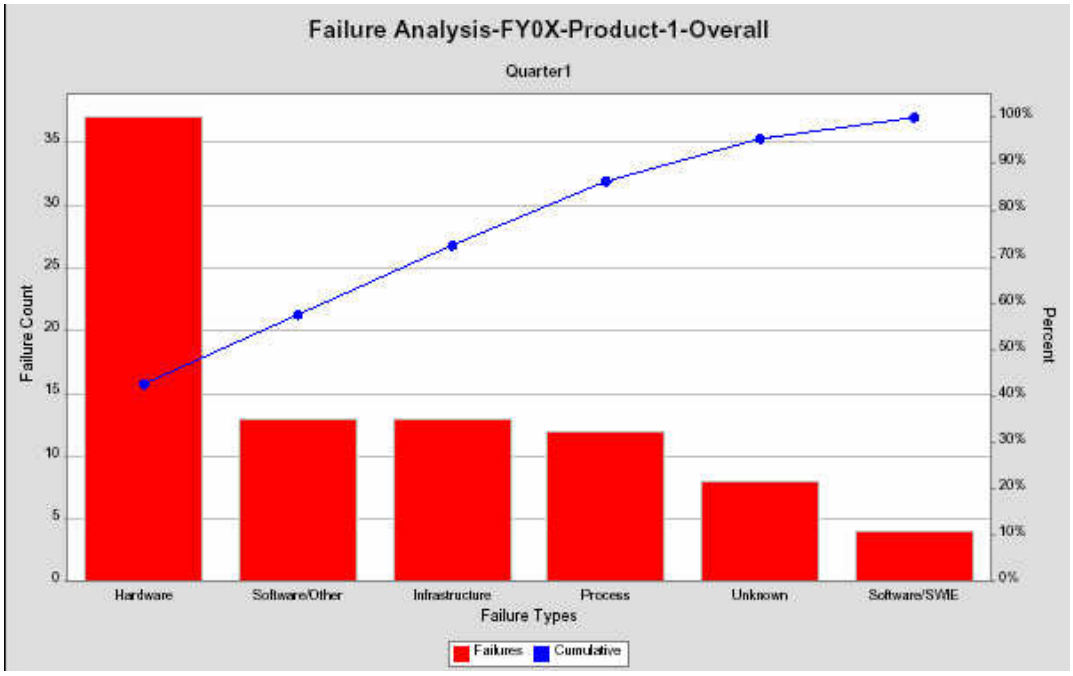


**Figure 16 - Failure Analysis Pareto chart**

Figure 17 shows the different types of failures categorized. The line curve embedded on the chart depicts a cumulative percentage of the total failures.

*CHAPTER 9*

# CONCLUSION

**"What did we do?"**

The Statistics and Metrics generator (SMG) produces web-based reports to provide feedback on performance and efficiency of a web based software download process. Performance metrics such as first pass percentage, overall yield percentage, failure numbers, volumes shipped are generated. It also generates execution metrics such as operating systems reports, Enterprise Standard Installation reports and domain reports. All reports are converted into PDF files because of its universal acceptance.

The failure analysis chart provides root cause, corrective action and the action required information. This provides feedback on why the installation failed enabling the management to drive business changes.

To build the system we did the following tasks:

- Existing data flow of the test process and its interaction with the software download process was studied. Information currently available, information required, data location, data retrieval process and data storage mechanism were understood.
- The log files produced during the test process and the data they contained were analyzed for reporting. A post parser was built to parse these log files and store the data in the Oracle(tm) database. The post parser was developed in Perl with Perl DBI for Oracle(tm) interface. It contained 600 lines of code.
- Software download specific data such as Operating system loaded, failure analysis, root cause and corrective action were found missing. New regular expressions were added to the post parser to gather this missing information in the database.
- Failure Analysis parser was built to monitor the software download process and report failure information. This parser was developed with "Expect" scripting utility. There

were about 60 different failure types. For each type of failure, 10 lines of code were written to implement the regular expression.

- The Oracle(tm) test process database that existed at Sun was used, with necessary modifications. A new table with multiple fields was added to this existing database to store software download specific data.
- A Java(tm) application was developed for producing reports in different chart formats. This application was built with Object Oriented design techniques. It used Java(tm) database connectivity for interfacing with the Oracle(tm) database. It included around 3000 lines of code.
- Jfree, a Java(tm) based reporting tool was used for producing graphically rendered charts in PDF format.
- Ant, a Java(tm) based build tool, was used to compile and build the code and generate the reports. A CVS repository was used for version control.
- A graphical user interface was developed for displaying the reports.

Thus the statistics and metrics generator provides a whole range of different reports and charts used for driving performance and efficiency. It also provides insight of what failures happened and how they can be rectified. These charts are extremely useful in driving process changes, clear software bugs, and communicating data in a consistent fashion of what happens in the test and software download process.

*CHAPTER 10*

# RECOMMENDATIONS FOR FUTURE WORK

- The Statistics and Metrics Generator right now generates static reports, in PDF format, based on queries written in a property file. We can also add a user interface to input requirements, build a dynamic query and run reports based on them. This would make the system more flexible, and more user oriented.

- The reports are developed in the form of PDF files looked up from web space. We can also display reports in the form the user wants it to be. Suggested formats might include HTML, Applet, Servlets and PDF. It can also be extended to Java(tm) Swing application.

- We can also build reports based on time such as Average time to complete software download, average time wasted on failures, cost graphs such as average cost for a time period of download, including resource and personnel costs.

- The system has been consciously developed to be able to be re-used at other external locations. The prototype built has been successful, but has not yet been ported outside of Sun.

# REFERENCES


[1] *Mass customization – B. Joseph Pine II*, Idea that drives existing web-based software installation architecture.

[2] Squid proxy web cache: http:// www.squid-cache.org

[3] Jfree, Free Java(tm) based reporting tool: http://www.jfree.org

[4] Apache Ant – Java(tm) based build tool: http://ant.apache.org

[5] Java(tm) Library for generation of PDF: http://www.lowagie.com/iText

[6] *Java2: The complete reference,* 3rd Edition, May 2000, By Patrick Naughton, Herbert Schildt, McGraw-Hill, ISBN 0-072-11976-4

[7] *Oracle Essentials*, 3rd Edition, June 2001, By Rick Greenwald, Robert Stackowiak, Jonathan Stern, O'Reilly, ISBN: 0-596-00179-7

[8] *Programming the Perl DBI*, Database programming with Perl, February 2000, By Alligator Descartes, Tim Bunce, O'Reilly, ISBN: 1-56592-699-4

[9] Open source development with CVS: http://cvsbook.red-bean.com

[10] Sun Internal websites: swie.west for process architecture, web-based installation background.

[11] *XML Developers Guide*, December 2000, By Fabio Arciniegas A., Fabio Arciniegas, McGraw-Hill, ISBN: 0-072-12648-5