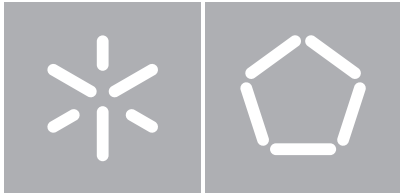




**Universidade do Minho**  
Escola de Engenharia



**Universidade do Minho**

Escola de Engenharia

Dissertação de Mestrado

# FITTEST logging and log-analysis' infrastructure for PHP

Luis Tiago Pereira

Thesis submitted to the University of Minho in the subject of  
Informatics, for the degree of Master of Science, under scientific  
supervision of Prof Dr. Carlos Ramalho and Prof Dr. Wishnu Prasetya

University of Minho

School of Engineering

Department of Informatics

January, 2014

# Resumo

*Muitas aplicações utilizam logs com o intuito de fornecer informações de funcionamento sobre as suas execuções. Geralmente, os logs gerados oferecem uma grande quantidade de informações que podem ser valiosas de inúmeras maneiras. Uma boa abordagem será um cenário onde o programa principal seja protegido contra quaisquer tipos de erros, tais como dados errados ou até mesmo nos piores cenários, ataques como a injeção de código (code injection). Atualmente, não existem soluções para apoiar todas estas características em uma única framework.*

*A abordagem apresentada para gerar estes logs é chamada de FITTEST que significa Future Internet Testing. FITTEST é um método de teste contínuo que foi escolhido para lidar com a dinâmica aumentada em aplicações de internet futuras. No entanto, para suportar a abordagem FITTEST, os logs têm de ser obtidos sistematicamente e em um formato bem definido.*

*Os logs FITTEST podem registrar dois tipos de logs: os eventos de alto nível e os eventos de baixo nível. Os de alto nível são eventos que podem ser vistos e produzidos pelo utilizador, enquanto que os de baixo nível são eventos que nos dizem o que aconteceu dentro de uma execução de uma função quando há uma reacção da aplicação para com um evento de alto nível.*

*O objectivo principal deste projeto é o desenvolvimento de uma infra-estrutura em PHP para gerar FITTEST logs, aplicar essa infra-estrutura em uma aplicação web e analisar os logs por ela gerados. Os logs são armazenados em um formato compacto (.log), no entanto podem ser exportados para XML possibilitando assim o pós-processamento por outras ferramentas.*



# Abstract

*Many applications employ logging in order to provide tracing information about their executions. Overall, generated logs offer a large amount of information that can be valuable in numerous ways. One good approach, will be a scenario where the main program is secure from any types of errors, such as wrong data or even in worst scenarios, things like code injection. Currently, there are no solutions to support all these features in a single framework.*

*The followed approach to generate these logs is FITTEST which means Future Internet Testing. FITTEST is an incessant testing method that has been chosen to handle the augmented dynamics in forthcoming internet applications. However, to support the FITTEST approach, logs have to be generated systematically, and in a well-defined format.*

*The logging solution can log both high level and low level events. High level events are events that can be seen as produced by users, whereas low-level events are events that tell us what happens inside a function execution as part of the target program's reaction to a high level event.*

*The goal of this project is the development of an infrastructure to generate FITTEST logs for PHP, the application of that infrastructure in a test web-application and then the analysing of the generated logs. The logs are then stored in a compressed format. However, they can also be exported to XML enabling post-processing by other tools.*



# Acknowledgements

*A challenge in which a successful outcome is assured isn't a challenge at all.*

*Jon Krakauer*

Along the performance of any work arise obstacles that would be difficult to overcome without the support of certain people; reminding us that the most important thing, is to never give up.

Each of these people makes me significant, but at the same time distinct. Although it is not possible to thank all of them, this is my sincerest thanks!

To my coordinators, Prof. Dr. José Carlos Ramalho and Prof. Dr. Wishnu Prasetya for their support, advice, recommendations, reviews and mainly for the time and attention contributed.

I thank my family for granting me the opportunity to attend the Master Degree and Erasmus Program; for simply always being there for me and constantly showing their support and dedication as a family.

To all my real friends for their positivity and understanding; and a special thank you to Sara Pereira, João Meira, Jennie Gil, and Claudio Brandão for their patience, assistance, and companionship throughout this project.





# List of Acronyms

ALE	Application Level Events
ALS	Analytic logging service
AOP	Aspect Oriented Program
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CPU	Central Processing Unit
CSS	Cascading Style Sheet
CSV	Comma Separated Values
DB	DataBase
DOM	Document Object Model
EFSM	Extended finite state machines
EPC	Electronic Product Code
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IIS	Internet Information Server
IP	Internet Protocol
IT	Information Technology
LTSV	Labeled Tab-separated Values
NCSA	National Center for Supercomputing Applications
ODBC	Open Database Connectivity
PDF	Portable document Format

REST	Representational State Transfer
RFID	Radio Frequency Identification
UTC	Coordinated Universal Time
W3C	World Wide Web Consortium
WAS	WebSphere Application Server
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

# Contents

<b>Resumo</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>List of Acronyms</b>	<b>vii</b>
<b>List of figures</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Technologies and tools</b>	<b>3</b>
2.1 MySQL . . . . .	3
2.2 PHP (Hipertext Preprocessor) . . . . .	3
2.3 JavaScript . . . . .	4
2.4 XML . . . . .	4
2.5 oXygen . . . . .	5
<b>3 State of the Art</b>	<b>7</b>
3.1 Logging Approach . . . . .	8
3.1.1 Type of Loggers . . . . .	9
3.1.2 Log Formats . . . . .	10
3.1.2.1 NCSA log format . . . . .	10
3.1.2.2 W3C Extended log format . . . . .	13
3.1.2.3 WAS log Format . . . . .	15
3.1.2.4 Apache Log4j . . . . .	17
3.1.2.5 Labeled tab-separated values . . . . .	19
3.1.2.6 Syslog . . . . .	20
3.2 Applications of Logs . . . . .	22
3.2.1 Applications areas . . . . .	22

---

3.2.2	Applications in Testing . . . . .	23
3.3	Alternative Approaches . . . . .	26
3.4	Web analytics . . . . .	26
3.4.1	History and evolution of web analytics . . . . .	26
3.4.2	Web server logging . . . . .	28
3.4.3	Tools and web applications . . . . .	29
<b>4</b>	<b>FITTEST log Format</b>	<b>39</b>
4.1	Syntax . . . . .	40
4.2	Base Tags . . . . .	41
4.2.1	Object Format . . . . .	41
4.2.2	Event Format . . . . .	43
4.3	FITTEST log file to XML log file . . . . .	46
4.4	XML log file . . . . .	47
<b>5</b>	<b>Test Example</b>	<b>51</b>
5.1	Repository Layout . . . . .	53
5.2	Projects . . . . .	54
5.3	Users . . . . .	57
5.4	Statistics . . . . .	60
5.5	Improvements . . . . .	61
<b>6</b>	<b>PHP Infrastructure</b>	<b>63</b>
<b>7</b>	<b>Analising the FITTEST logs</b>	<b>67</b>
<b>8</b>	<b>Conclusion</b>	<b>77</b>
	<b>Bibliography</b>	<b>83</b>

# Figures

2.1	Functioning of php pages . . . . .	4
3.1	WebTrends Analytics collects and analyzes data . . . . .	31
3.2	Omniure Analytics . . . . .	32
3.3	Rapid data discovery for very large datasets . . . . .	33
3.4	Coremetrics Analytics . . . . .	34
3.5	Urchin dashboard . . . . .	35
3.6	Piwik dashboard . . . . .	37
5.1	OAIS model . . . . .	51
5.2	Test Application (Repository architecture). . . . .	52
5.3	Test Application (Repository online). . . . .	53
5.4	Test Application (Access Denied). . . . .	54
5.5	Test Application (Upload Page). . . . .	54
5.6	Test Application (Upload Form). . . . .	55
5.7	Test Application (Warning message). . . . .	56
5.8	Test Application (Project List). . . . .	56
5.9	Repository HTML tree . . . . .	57
5.10	Test Application (Project Detail). . . . .	58
5.11	Test Application (Popup Detail). . . . .	58
5.12	Test Application (User List). . . . .	59
5.13	Test Application (User Form). . . . .	59
5.14	Test Application (Error Message). . . . .	59
5.15	Test Application (Success Message). . . . .	60
5.16	Test Application (Statistics page). . . . .	61
5.17	Test Application (Chart page). . . . .	61
7.1	FITTEST Web Application. . . . .	71
7.2	Answers overview . . . . .	72
7.3	User types Groups . . . . .	72

---

7.4	Categories overview . . . . .	73
7.5	List of vulnerabable pages . . . . .	73
7.6	Post-state by groups . . . . .	74
7.7	Post-state of all events . . . . .	74
7.8	Logs Editor . . . . .	75

# Chapter 1

## Introduction

Many applications employ logging in order to provide tracing information about their executions. The generated logs provide a lot of information that can be useful in many ways, e.g. for producing usage statistics, to be analysed to help us diagnosing errors, or even to directly detect errors[1].

Code coverage tools and profilers can use logs to determine which part of the code is executed and the resource consumption during an execution. Some also try to use them to infer malicious activities and security-policy violations[1].

An application can register a wide range of events into a log. This can be anything from logging notable activities, irregular events, or logging unexpected errors. The 'events' are categorized into two types.

The first type covers application events, also called high-level events. These events are typically user to represent user interactions with the target application (or in general, interactions of other types of external agents), such as when a user click on a button.

The second are low-level events and they are typically used to represent events occurring within the application itself, such as when some internal method/function `f` is being called.

We call the logging of lower level events deep logging. The logging framework can log both types of events.

Event Logs play a very crucial role in modern days IT systems. Since Event logs are collected in real time by the system components, they serve as an excellent source of information to monitor systems and overall networks. Apart from this, the information obtained from event logs are used in log analysis which is very helpful during



audit procedure and management of IT infrastructure.

Log analysis has become an important source for network security and administration. The goal of web analysis is to offer some help to network administrators and server managers, who handle the complex responsibility of diagnosing unusual system and network behavior by decoding the vital information present in the event log.

Most work in logging has been focused on providing logging infrastructure for various software technologies. Many modern programming languages already come with logging libraries. These provide functions that we can use to log messages. They often have a notion of 'logging level' to control the verbosity of the generated logs.

This thesis is divided into six chapters. Chapter 2 explains the technologies and tools that support the work. Chapter 3 is related to the State of the art, will be addressed some logging approach, the different types of logs formats and presented the main web analysis applications. Chapter 4 introduces the FITTEST log format, its syntax and event types and is followed by Chapter 5 that show a web page used to make some tests. Chapter 6 discusses the PHP infrastructure for the log production and the last chapter explains the application of data mining on logs and answers the research questions.

# Chapter 2

## Technologies and tools

This chapter consists in the discussion of the technologies and tools that supported the development of this work. To develop the web application and infrastructure for logging it is necessary to have some knowledge of PHP and XML. The Javascript language is not mandatory, but can be used to help to create more dynamic pages.

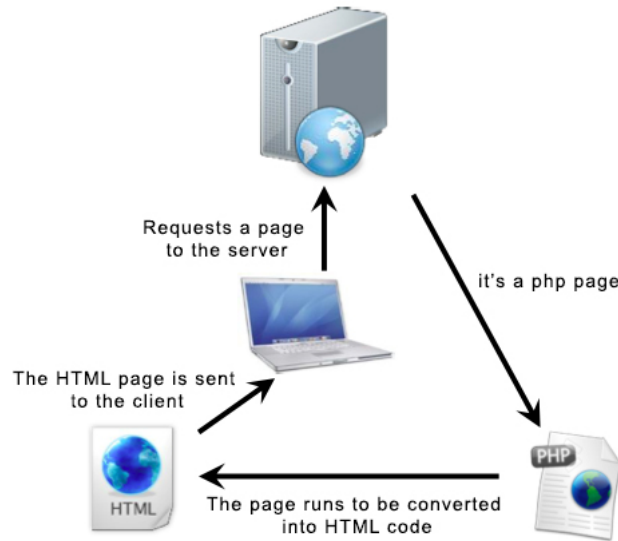
### 2.1 MySQL

MySQL is a relational database management system (RDBMS) which supports the language SQL (Structured Query Language) for handling the data. A key feature of MySQL is the fact that this is developed in open source and work with a large number of operating systems: Windows, Linux, FreeBSD, BSDI, Solaris, Mac OS X, SunOS, SGI, etc. Is renowned for its performance, ruggedness and also for being multi-tasking and multi-user. [3]

### 2.2 PHP (Hipertext Preprocessor)

PHP is a programming language that allows to create dynamic Web sites enabling interaction with the user through forms, URL parameters, and links. The difference of PHP over other similar languages, such as Javascript is that the PHP code is executed on the server and sent to the client only HTML. This way possibilities the interaction with databases and existing applications on the server, with the advantage of not exposing the source code to the client. This can be quite useful when working with keywords or any other kind of confidential information. Figure 2.1 shows the functioning of PHP pages. [4, 5]

Figure 2.1: Functioning of php pages



## 2.3 JavaScript

Javascript is a programming language used to create small programs (scripts) in charge of performing actions in the context of a web page. This is a programming language from the client side, because it's the browser that supports its processing. Thanks to its compatibility with the majority of Browsers is the programming language more used by the client. With Javascript can be create special effects on the pages and set of interactive activities with the user. The browser is in charge of interpreting Javascript instructions and run them to perform these effects and interactive activities, so that the greatest resource, and perhaps the only, with which this language is the browser itself. [6]

## 2.4 XML

XML, (eXtensible Markup Language), XML, is a markup language for creating documents with data organized hierarchically such as texts, databases or vector. The language is classified as XML Extensible because it allows the definition of the markup elements. XML provides a basic syntax that can be used to share information between different computers and applications. When combined with other patterns, makes it possible to define the content of a document separately from its format, making it simple to reuse the code in other applications for different purposes. [7]

## 2.5 oXygen

oXygen XML Editor is the complete XML development and authoring platform. It is a Java application, so it can run in windows, Mac OSX and Linus. It provides the must have tools for working with a wide range of XML standards and technologies. oXygen XML Editor includes both the development features of oXygen XML Developer and the authoring features of oXygen XML Author, as well as the diff and merge functionality of oXygen XML Diff. [8]

These are the technologies that will be used for the development of this work. In all of them was used the latest version available. The operating system used was the MAC-OSX and the toolkit for web was MAMP.



# Chapter 3

## State of the Art

Information systems, nowadays, play a large role within organizations in supporting business processes. Typically, all events that are handled by the system are recorded. These records of activities handled by the information system are called event logs. In general an event log records the events that occur in a certain process for a certain case.

These event logs are basically collected information about system behavior. There are different groups of people involved in the software development process like system administrators, developers, testers who would like to know what has been going on with the system over its life time and with that use logs for their own benefits:

1. Developers: debugging (to provide execution traces and actual failure), profiling;
2. Testers: failure detection (to follow the program logic without having to run it);
3. Performance analyst: profiling (to identify bottlenecks and increase performance);
4. Support engineers: failure detection (to analyze what users did);
5. Web masters and system administrators: configuring or tuning the system;
6. Marketers: collecting different statistics.

The information can be expressed in different formats and these formats are often determined by the choice of automatic analysis tools. Often, all those tools

are developed to serve a specific user group and their needs. People may also use different terminologies, e.g. developers use the term tracing instead of logging, when they are dealing with debugging issues. It is very important and necessary to identify failures or to show their absence. Testing teams work on that and sometimes it's very difficult to identify them without involving formal methods instead of testing. This is a case where the use of logs can be advantageous to test teams. It is such a log format and an application of logs which this report is mainly about.

### 3.1 Logging Approach

From the most general perspective a log is just a plain text file, where every live record represents either abstraction of an event in the system, or the system's internal state at a concrete moment, or both. Usually all records are tagged with time stamps and have types from a list of predefined types for a given application. In computing, an event is an action or occurrence detected by the program that may be handled by the program. Typically events are handled synchronously with the program flow, this means, the program has one or more dedicated places where events are handled, frequently an event loop. An event is represented in computer systems by an object that contains data about the event, such as the time it happened, the location, what other events caused it to happen, and many other attributes. *System-level events* and *application-level events (ALE)* are the two conceptually different categories of events there exists.

*System-level events* as method calls, object creation, exception throwing and variable assignments. *Application Level Events (ALE)* is a standard created by Electronic Product Code (EPC) - Global, Inc., an organization of industry leaders devoted to the development of standards for the Electronic Product Code (EPC) and Radio Frequency Identification (RFID) technologies. The ALE specification is a software specification indicating required functionality and behavior, as well as a common API expressed through XML Schema Definition (XSD) and Web Services Description Language (WSDL). This category includes high-level events, which are involved in the definition of the application business logic. Examples of such events can be user registration, buying items, charts generation and the events work-flow associated with them as well as Graphical User Interface (GUI) interaction events.

### 3.1.1 Type of Loggers

Log an application events can actually be done by many different ways. Use a *print*-like function at places where we want to log is the easiest solution. The function can only write the text version of its arguments into the log file. Despite its simplicity and low-levelness, *print* proved to be a very useful technique in debugging and underlies many logging frameworks. *in-code logger* are the name called on loggers based on this simple *print* approach and typically they are easy to use and understand. However, for instance it requires lots of duplications and overwork when we need to log multiple function calls. This approach suffers from one big disadvantage: put to much additional work on the developers, and therefore can be error prone because of the human factor.

The solution for this disadvantage is based on the idea of aspect-oriented programming (AOP). For performance, AOP approach certainly has a little overhead against the traditional one. One of the many strengths of AOP is that it allows to separate non-business concerns from business logic. It also helps in mundane tasks such putting a logging logic in each of the methods. Logging statements are written as a separate concern, as an aspect, which are then weaved into the application's code. Loggers built like this are called *aspect-oriented loggers*. Such an approach is however often hard to implement and is not easy to use. This may be the reason why it is not widely used. In addition, the number of operations supported by the join-point model of the used AOP framework may be limited.

The aspect-oriented approach, more generally, is an injection approach, where a set of separately defined logging statements are injected into the application's code. Within this class we can distinguish between source code and byte-code injection approach. Build loggers with byte-code are called *byte-code level loggers*. To inject log statements to an already compiled application is used binary instrumentation and there are two types of injection: static and dynamic. The static means that the injection is done at the compile time, before the application is deployed whereas the dynamic means that the injection is decided and done while the application is running.

In any approach we can't provide a guarantee that the added logging statements do not change the intended behavior of the original program. A simply *print* statement, for example, can change the timing behavior and hence change the original



behavior. It may also throw an exception.[2]

### 3.1.2 Log Formats

Logging an event should be a lightweight activity, so that the act of logging does not itself slow down the program enough to impact its operation or throughput. And writing to a debugging log should not slow down the program enough to mask (or introduce) subtle timing problems that would (or wouldn't) show up in release builds.

The fastest type of logging would be writing text to an in-memory buffer, and then flushing that data to disk later when the program is otherwise quiescent. But with this solution, any unflushed data will be lost if the program crashes. This logfile would be an unreliable record of what led up to the crash.

Because of those constraints, most logging systems simply append log events to the end of the existing logfile and then flush or close it. These constraints would seem to rule out a fully-enclosed format such as standard XML, since the standard XML libraries cannot simply append a data item to a XML file without first reading in the whole XML file so that they can build up the complete DOM (Document Object Model) tree.

To simplify or rather obstruct the subsequent analysis phase is built on the choice of the output format of logs. It should encompass sufficient information for the analysis, but meanwhile not be too verbose. Below will be shown some examples of existing log formats, which are commonly used in real systems.

#### 3.1.2.1 NCSA log format

NCSA (National Center for Supercomputing Applications) is a very popular format, supported by many servers. We can split NCSA log format in 3 types: The NCSA common log, NCSA combined and separate with date.

The NCSA common log file format is a fixed ASCII text-based format, so it can not be customize. The NCSA Common log format contains only basic HTTP access information. The NCSA Common Log, sometimes referred to as the Access Log, is the first of three logs in the NCSA Separate log format. The Common log format can also be thought of as the NCSA Combined log format without the referral and user agent. [14]

The Common log contains the requested resource and a few other pieces of information, but does not contain referral, user agent, or cookie information. The information is contained in a single file and the fields are:

host	rlogn	username	date:time	request	statuscode	bytes
------	-------	----------	-----------	---------	------------	-------

The following example shows these fields populated with values in a common log file record:

125.125.125.125 - dsmith [13/Oct/1999:14:25:02 +0500] "GET /index.html HTTP/1.0" 200 1043
--

The following is a description of the fields in the common log format:

- **host** (125.125.125.125 in the example) The IP address or host/subdomain of the HTTP client that made the HTTP resource request.
- **rlogn** ("- " in the example) Remote log name.
- **username** (dsmith in the example) The username, (or user ID) used by the client for authentication.
- **Date:time timezone** (13/Oct/1999:14:25:02 +0500 in the example) The date and time stamp of the HTTP request.
- **request** (GET /index.html HTTP/1.0 in the example) The HTTP request. The request fields contains three pieces of information. The main piece is the requested resource (index.html).The request fields also contains the HTTP method (GET) and the HTTP protocol version (1.0)
- **statuscode** (200 in the example) The status is the numeric code indicating the success or failure of the HTTP request. In this case is a success message.
- **bytes** (1043 in the example) The bytes field is a numeric field containing the number of bytes of data transferred as part of the HTTP request, not including the HTTP header.

The NCSA combined log format is an extension of the NCSA common log format. The combined format contains the same information as the common plus three (optional) additional fields: the referral field, the useragent field, and the cookie field.

The following are the fields in Combined log format (the second line):

host	rlogn	username	date:time	request	statuscode	bytes
referrer	user\_agent	cookie				

Example of the Combined log format:

```
125.125.125.125 - dsmith [13/Oct/1999:14:25:02 +0500]
"GET /index.html HTTP/1.0" 200 1043 "http://www.ibm.com/"
"Mozilla/4.05 [en] (WinNT; I)" "USERID=CustomerA;IMPID=01234"
```

The following are description of the three additional fields:

- **referrer** ("http://www.ibm.com/" in the example) The URL which linked the user to your site
- **user\_agent** ("Mozilla/4.05 [en] (WinNT; I)" in the example) The web browser and platform used by the visitor to your site
- **cookie** ("USERID=CustomerA;IMPID=01234" in the example) Cookies are pieces of information that the HTTP server can send back to client along the with the requested resources. Cookies take the form `KEY = VALUE`. Multiple cookie key-value pairs are delineated by semicolons(;).

The NCSA Separate log format, sometimes called three-log format, refers to a log format in which the information gathered is separated into three separate files (or logs), rather than a single file. The three logs are often referred to as: Common log or access log; Referral log; and Agent log.

The three-log format contains the basic information in the NCSA common log format in one file, and referral and user agent information in subsequent files. However, no cookie information is recorded in this log format.

The first of three logs is Common log, sometimes referred to as the access log, which is identical in format and syntax to the NCSA Common log format.

The referral log are the second of the three logs. The referral log contains a corresponding entry for each entry in the common log. The fields in Referral log are only two:

```
date:time    referrer
```

The following is an example of a record in a referral log:

```
[13/Oct/1999:14:25:02 +0500] "http://www.ibm.com/index.html"
```

The following is a description of the fields in the Referral log:

- **date:time timezone** ([13/Oct/1999:14:25:02 +0500] in the example) The date and time stamp of HTTP request. The date and time for an entry logged in the referral log corresponds to the resource access entry in the common log. The syntax of the date stamp is identical to the date stamp in the common log.
- **referrer** ("http://www.ibm.com/index.html" in the example) The referrer is the URL of the HTTP resource that referred the user to the resource requested. For example, if a user is browsing a web page such as *http://www.ibm.com/index.html* and the user clicks on a link to a secondary page, then the initial page has referred the user to the secondary page. The entry in the referral log for the secondary page will list the URL of the first page (*http://www.ibm.com/index.html*) as its referral.

The Agent log is the third of the three logs making up the three-log format. Like the referral log, the agent log contains a corresponding entry for each entry in the common log. The fields in the Agent log are:

<code>date:time</code>	<code>agent</code>
------------------------	--------------------

The following is an example of a record from an Agent log:

<code>[13/Oct/1999:14:25:02 +0500] "Microsoft Internet Explorer - 5.0"</code>
---

The following is a description of the fields in the Agent log:

- **date:time timezone** ([13/Oct/1999:14:25:02 +0500] in the example) The date and time stamp of HTTP request.
- **agent** ("Microsoft Internet Explorer - 5.0" in the example) The HTTP client that makes HTTP request. It is customary for an HTTP client, such as a web browser, to identify itself by name when making an HTTP request. It is not required, but most HTTP clients do identify themselves by name. The Web server writes this name in the agent log.

### 3.1.2.2 W3C Extended log format

Most Web servers offer the option to store logfiles in either the common log format or a proprietary format. The common log file format is supported by the majority of analysis tools but the information about each server transaction is fixed. In many cases it is desirable to record more information. Sites sensitive to personal

data issues may wish to omit the recording of certain data. In addition ambiguities arise in analysing the common log file format since field separator characters may in some cases occur within fields. The extended log file format is designed to meet the following needs:

- To permit control over the data recorded.
- To support needs of proxies, clients and servers in a common format
- To provide robust handling of character escaping issues
- To allow exchange of demographic data.
- To allow summary data to be expressed.

The W3C (World Wide Web Consortium) Extended log file format is the default log file format used by ISS (Microsoft Internet Information Server).

A log file in the extended format contains a sequence of lines containing ASCII characters. Each line may contain either a directive or an entry. Entries consist of a sequence of fields relating to a single HTTP transaction. Fields are separated by white space. If a field is unused in a particular entry dash "-" marks the omitted field. Directives record information about the logging process itself. [15, 16]

Lines beginning with the # character contain directives. The following directives are defined:

- **Version** <integer>.<integer> The version of the extended log file format used. This draft defines version 1.0.
- **Fields** [<specifier>...] lists a sequence of field identifiers specifying the information recorded in each entry.
- **Software** string Identifies the software which generated the log.
- **Start-Date** <date> <time> The date and time at which the log was started.
- **End-Date** <date> <time> The date and time at which the log was finished.
- **Date** <date> <time> The date and time at which the entry was added.
- **Remark** <text> Comment information. Data recorded in this field should be ignored by analysis tools.

The following is an example of a record in the extended log format that was produced by the Microsoft Internet Information Server (IIS):

```
#Software: Microsoft Internet Information Server 6.0
#Version: 1.0
#Date: 1998-11-19 22:48:39
#Fields: date time c-ip cs-username s-ip cs-method cs-uri-stem
cs-uri-query sc-status sc-bytes cs-bytes time-taken cs-version
cs(User-Agent) cs(Cookie) cs(Referrer)
```

The directives Version and Fields are required and should precede all entries in the log. The Fields directive specifies the data recorded in the fields of each entry.

The following prefixes and identifiers do not require a prefix:

Table 3.1: Prefix

Prefix	Meaning
s-	Server actions.
c-	Client actions.
cs-	Client-to-server actions.
sc-	Server-to-client actions.

### 3.1.2.3 WAS log Format

WebSphere Application Server (WAS) provides an API for application level logging called Analytic logging service (ALS). We can use this API from our own web application. The IBM WebSphere Personalization server uses this API to log data.

This API supports three logging methods: HTTP, database, and file. The record format for logged data differs slightly based on the logging method used; however, each log record contains the following five fields:

```
appid reqid httpdata cookie appdata
```

Example of WAS log:

```
<custCareID> <01010123-1234> <-> <-> <target=TP600E;crosssells=
TP770,TP380;ads=pIII>
```

The following is a description of these fields:

- **appid** (<custCareID> in the example) The Application ID of the record. Appid allows multiple applications to log to the same medium.

Table 3.2: Identifiers do not require a prefix

Field	Appears As	Description
Date	date	The date that the activity occurred.
Time	time	The time that the activity occurred.
Client IP Address	c-ip	The IP address of the client that accessed your server.
User Name	cs-username	The name of the authenticated user who accessed your server.
Service Name	s-sitename	The Internet service that was accessed by a client.
Server Name	s-computername	The name of the server on log entry was generated.
Server IP Address	s-ip	The IP address of the server on which the log entry was generated..
Server Port	s-port	The port number the client is connected to.
Method	cs-method	The action the client was trying to perform.
URI Stem	cs-uri-stem	The resource accessed.
URI Query	cs-uri-query	The query, if any, the client was trying to perform.
Protocol Status	sc-status	The status of the action, in HTTP or FTP terms.
Win32 Status	sc-win32-status	The status of the action.
Bytes Sent	sc-bytes	The number of bytes sent by the server.
Bytes Received	cs-bytes	The number of bytes received by the server.
Time Taken	time-taken	The duration of time that the action consumed.
Protocol Version	cs-version	The protocol (HTTP, FTP) version used by the client.
Host	cs-host	Displays the content of the host header.
User Agent	cs(User-Agent)	The browser used on the client.
Cookie	cs(Cookie)	The content of the cookie sent or received, if any.
Referrer	cs(Referer)	The previous site visited by the user.

- **reqid** (<01010123-1234> in the example) lists a sequence of field identifiers specifying the information recorded in each entry.
- **httpdata** (<-> in the example) The NCSA Combined log. The fields in the log are host rlogn username date request statuscode bytes referrer user\_agent and cookie.
- **cookie** (<-> in the example) The date and time at which the log was started.
- **appdata** <date> <time> The date and time at which the log was finished.
- **Date** (<target=TP600E;crosssells=TP770,TP380;ads=pIII> in the example) The application data. The appdata field can contain any application level data. The format of the APPDATA field is a list of semicolon-separated key=values pairs, where key represents the semantic label of the values, and the values is a list of comma-separated tags or product identifiers. If a comma is part of the value, quotes are used to enclose the value.

### 3.1.2.4 Apache Log4j

Log4j [17, 18] provide, e.g., improved APIs or improved performance. Apache log4j is a logging library for java. Apache Commons Logging offers a set of common logging APIs so that the implementation of the logger can be decoupled and replaced easily.

The following table defines the log levels and messages in log4j, in decreasing order of severity. The left column lists the log level designation in log4j and the right column provides a brief description of each log level.

Table 3.3: Log4j log level

Level	Description
OFF	The highest possible rank and is intended to turn off logging.
FATAL	Severe errors that cause premature termination. Expect these to be immediately visible on a status console.
ERROR	Other runtime errors or unexpected conditions. Expect these to be immediately visible on a status console.
WARN	Use of deprecated APIs, poor use of API, 'almost' errors, other runtime situations that are undesirable or unexpected, but not necessarily "wrong". Expect these to be immediately visible on a status console.
INFO	Interesting runtime events (startup/shutdown). Expect these to be immediately visible on a console, so be conservative and keep to a minimum.
DEBUG	Detailed information on the flow through the system. Expect these to be written to logs only.
TRACE	Most detailed information. Expect these to be written to logs only.

There are three ways to configure log4j: with a properties file, with a XML file and through Java code. Within either can be defined three main components: Loggers, Appenders and Layouts.

Loggers are logical log file names. They are the names that are known to the Java application. Each logger is independently configurable as to what level of logging (FATAL, ERROR, etc.) it currently logs. The actual outputs are done by Appenders. There are numerous Appenders available, with descriptive names, such as FileAppender, ConsoleAppender, SocketAppender, SyslogAppender, NTEvent-LogAppender and even SMTPAppender. Appenders uses Layouts to format log entries. A popular way to format one-line-at-a-time log files is PatternLayout, which uses a pattern string, much like the C / C++ function printf.

Configuring logging via a file has the advantage of turning logging on or off without modifying the application that uses log4j. The application can be allowed to run with logging off until there's a problem, for example, and then logging can be turned back on simply by modifying the configuration file.



Following is a simple configuration file for PatternLayout:

```
# Define the root logger with appender file
log = /usr/home/log4j
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern=
    %d{yyyy-MM-dd}-%t-%x-%-5p-%-10c:%m%n
```

Following is a Java Example which would generate logging information:

```
import org.apache.log4j.Logger;

import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class log4jExample{
    /* Get actual class name to be printed on */
    static Logger log = Logger.getLogger(
        log4jExample.class.getName());

    public static void main(String[] args)
        throws IOException,SQLException{

        log.debug("Hello this is an debug message");
        log.info("Hello this is an info message");
    }
}
```

Compile and run the above program, it would create a log.out file in /usr/home/log4j directory which would have following log information:

```
2010-03-23-main--DEBUG-log4jExample:Hello this is an debug message
2010-03-23-main--INFO -log4jExample:Hello this is an info message
```

### 3.1.2.5 Labeled tab-separated values

Labeled Tab-separated Values (LTSV) [19] format is a variant of Tab-separated Values (TSV). Each record in a LTSV file is represented as a single line. Each field is separated by TAB and has a label and a value. The label and the value have been separated by ':'. With the LTSV format, you can parse each line by splitting with TAB (like original TSV format) easily, and extend any fields with unique labels in no particular order.

Recently the importance of log analysis is increasing, so it is often added to logs a custom values not contained by a standard format, or sometimes it is forced to output logs which are frequently ordered to change its format. LTSV is suitable for these cases, too.

The LTSV format originally focuses on access logs of web servers, so we can see an access log of traditional Combined Log Format and the same log of LTSV format version as examples. Following is the configuration of LTSV format:

```
LogFormat "%h\tident:%l\tuser:%u\ttime:%t\treq:%r\tstatus:
%>s\tsize:%b\treferer:%{Referer}i\tua:%{User-Agent}i" combined_ltsv
```

The access log is look like:

```
host:127.0.0.1<TAB>ident:-<TAB>user:frank<TAB>time:
[10/Oct/2000:13:55:36 -0700]<TAB>req:GET /apache_pb.gif
HTTP/1.0<TAB>status:200<TAB>size:2326<TAB>referer:
http://www.example.com/start.html<TAB>ua:Mozilla/4.08 [en]
(Win98; I ;Nav)
```

As we can see the produced log is difficult to read, but applying a simple LTSV parser:

```
#!/usr/bin/env ruby

while gets
```

```

record = Hash[$_.split("\t").map{|f| f.split(":", 2)}]
p record
end

```

the result is look like this:

```

{"host"=>"127.0.0.1", "ident"=>"-", "user"=>"frank", "time"=>
"[10/Oct/2000:13:55:36 -0700]", "req"=>"GET /apache_pb.gif HTTP/1.0",
"status"=>"200", "size"=>"2326", "referer"=>"http://www.example.com/
start.html", "ua"=>"Mozilla/4.08 [en] (Win98; I ;Nav)\n"}

```

### 3.1.2.6 Syslog

Syslog [20] is a standardized mechanism for logging in computer systems and was developed as a part of the sendmail project. It became the standard logging solution on Unix-like systems and other operating systems. It permits separation of the software that generates messages from the system that stores them and the software that reports and analyzes them. It also provides a feature to notify administrators of problems or performance. Syslog log entries represent information about activities of different system components (demons, kernel, ftp, user and so on) with assigned priority/levels (Alert, Critical, Warning, Error, Message and so on). For more deeply, the log messages are given two parameters: A level of severity and a 'facility' - category that describes the part of the system that generated the log message. Both parameter values are decisions made by each programs' designers.

The severity levels:

- emerg - 'panic' situations
- alert - urgent, less than panic, situations
- crit - 'critical' conditions
- err - error conditions
- warning - warning (non-error) messages
- notice - things that may require attention
- info - informational messages
- debug - debugging messages

The standard facilities:

- kern - the kernel
- user - "user" processes (no specific)
- mail - sendmail
- daemon - "system" daemons, such as 'routed'
- auth - security and authorization-related
- lpr - BSD line-printer daemon
- news - usenet news system
- uucp - for the (ancient) UUCP service
- cron - the cron daemon
- syslog - used by syslogd itself
- ftp - for ftp services
- authpriv - similar to 'auth', logged to secure file
- local0 - local7 - used for local/other daemons

Syslog is implemented in two parts, one part is a set of (standard) C-library routines: read the section-3 manpage 'syslog' for details (programs utilize these routines to send messages to the log daemon) and the other part is a daemon process (syslogd): The log daemon runs all the time, Writes log messages to various locations and configure with the /etc/syslog.conf file.

#### Example of Syslog Output

```
Mar 12 12:00:08 server2 rcd[308]: Loaded 12 packages in 'ximian-red-
carpet|351' (0.01878 seconds)
Mar 12 12:00:08 server2 rcd[308]: id=304 COMPLETE 'Downloading
https://server2/data/red-carpet.rdf' time=0s (failed)
Mar 12 12:00:08 server2 rcd[308]: Unable to download licenses
info: Unable to authenticate - Authorization Required
(https://server2/data/red-carpet.rdf)
Mar 12 12:10:00 server2 /USR/SBIN/CRON[6808]: (root) CMD
( /usr/lib/sa/sa1 )
Mar 12 12:20:00 server2 /USR/SBIN/CRON[6837]: (root) CMD
( /usr/lib/sa/sa1 )
```

## 3.2 Applications of Logs

### 3.2.1 Applications areas

There are several areas of applications where logs are used widely:

1. Data Bases: logs produced are called *transaction logs* or *database logs*;
2. Operating systems: logs produced are called *system log files*;
3. Web application and servers: produce the most widespread type of logs called *web-logs*;
4. Distributed system;
5. Supercomputing;

In *transaction logs* or *database logs* a log represents a history of updates done to the database and can be used to recover the database state in the case of crashes or hardware failures, thus guaranteeing transactions reliability. These logs are binary (not human-readable).

Windows Event Viewer [21] and Unix System Activity Reporter use the *system log files* for performance monitoring and debugging. These logs typically record device changes, device operations, security relevant events and can be processed by other tools for various purposes.

The *web-logs* logs can be used by system administrators, performance engineers and even marketers. They store information about a server and its clients. Typically web logs are used to get an overview of requested pages history and including many useful information such as client IP address, request date/time, HTTP response code and so on.

In Distributed systems logging has been applied to facilitate debugging because Distributed systems are usually complicated due their synchronization and non-determinism issues. A framework proposed in '*Formal Modeling of Communication Traces*' [22] applies instrumentation, and uses communicating automata for modeling the generated event traces.

In the field of supercomputing, systems are often a complex composition of different

software and hardware. A good technique for failure prediction is highly desirable and logs can be a good source of information to do it. Currently, the massive size, complexity, and lack of standard formats in supercomputer's logs make them difficult to analyze.

The work done in '*Extracting the textual and temporal structure of supercomputing logs*' [23] proposes a solution by using textual clustering to build the syntactic structures of log messages and then classify messages into semantic groups, taking advantage of online clustering algorithms. The work done in '*Bluegene/l failure analysis and prediction models*' [24] studies BlueGene/l execution logs to infer some laws on system observations; these are exploited to do failure prediction.

### 3.2.2 Applications in Testing

Over the years there have been made many researches on log files and logs analysis approach. Bellow we can see these researches and find the importance of application of logging.

#### Failure Analysis and Detection

In '*Testing using log file analysis: tools, methods and issues*' [25] and '*Theory and practice of log file analysis*' [26] the author of the work (Andrews) gave a log analyzer specification language(LFAL), which is a formal way to represent a set of state machines whose transitions are labelled by entries from a log file. So, such a machine specifies how to analyze the file, and is used to decide whether the log is accepted or to reveal an error. With the framework two case studies were included: unit testing and system testing. Also, the author gave an extended review on log-based analyses and introduced a set of terminologies for the field. Later on '*Broad-spectrum studies of log file analysis*' [27], '*Industrial evaluation of a log file analysis methodology*' [28] and '*A Method of Log File Analysis for Test Oracle*' [29] the approach was extended by requirements clarification and random test-cases generation.

Even if we know that a log contains evidence of a failure at some point, it can still be very hard for the testing team to identify the cause of the failure, especially if the log contains a lot of events. This can happen when an application has been running for quite some time when the failure happens. This problem was studied in '*Investigation of failure causes in workload-driven reliability testing*' [30]. The solution proposed is based by comparing the log that exhibits the failure and the application's behavior model inferred from previous log files.

## Failure Analysis and Detection

Another research direction where analysis of log files was considered useful is requirements monitoring. The first introduction of this approach was in 1998 on '*Rapid application of lightweight formal methods for consistency analyses*' [31] and 1999 on '*Automatic generation of test oracles*' [32] where databases was exploited through analysis engine. The analysis process consisted of the five steps. First prepare a database to able to store logging information, then load the information in the database, third step is convert requirements to DB queries and at last interpret the results. This approach and Andrews' work on LFAL [25] were mentioned in '*Test oracles*' [33] as prospective future approaches on the oracle problem. The approaches are general in the sense that they require neither precomputed inout/output pairs nor a previous version of the system under test.

Another development in requirements monitoring was '*Monitoring software requirements using instrumented code*' [34], which was also based on the application of logs but in a completely different way. The proposed framework can be used for continuous requirements analysis during the runtime and combines assertions checking and model checking to inform the monitor. The target program is instrumented in order to produce a stream of events, which can be translated to a sequence of method calls and used to generate a state-based model. The resulting model is checked for requirements violations and corresponding path is reported when one is found.

TestLog framework was proposed in '*Object-oriented legacy system trace-based logic testing*' [35], which for expressing test-cases uses execution traces as a basis. The traces are represented in the form of Prolog logic facts and a test-case is expressed in the form of a Prolog logic query. Thereby it eliminates the need to programmatically bring the system to a particular state, and provides the test-writer an abstract mechanism to query the traces.

## Security Testing

A complete history showing the usage of an application can be recorded by unceasingly monitoring and logging into the application's activities; which in turn is beneficial for internet security purposes. In addition, these log histories can be subjected to security analysis for further examination. For instance, E.g. Kowalsi et al. [36] in '*Improving security through analysis of log files intersections*' did this to distinguish intrusions. Kowalsi measured the connection of the web server's firewalls' log files and the server's access logs to make the study practical. By using

this method, potential attackers' IP addresses can be recognized.

### **Invariant/Specification Inference**

In order to gather and conclude information for program invariants, the use of invariant/specification inference log files is essential. An implement used for determining invariants within execution traces is Daikon in '*Dynamically discovering likely program invariants to support program evolution*'[37]. This process consists of the program being instrumented to trace pertinent variables and then run through several test-cases.

Candidate's invariants (in terms of both instrumented and derived variables) are systematically build, refined, and checked against the traced values. The invariants reported to the programmer are established by having passed the test with a sufficient number of positive witnesses. Evaluating information provided by static analysis, e.g. as in '*Automatic generation of program specifications*'[38], ensures the opportunity of methods like Daikon, to be reinforced.

In cases where memory and CPU resources are restricted, proper examination of log files can be very useful for real-time critical embedded applications; this has been proven by a recent work '*Formal analysis of log files*'[39] in the area of runtime verification. The author practices sequential reasoning for identifying log properties. When evaluating logs, is it vital for data-parameterization to be supported by the specification language (events typically carry data). Specified properties, called patterns, are interpreted to data-parameterized automatic and conveyed in a division of the recorded RULER language. The greater chance of expressive automata in a specification, the more combinations of patterns can be created by users.

Parameterized machines are envisioned using the GRAPHVIZ tool. Additionally, the system offers preliminary support for easily understanding specification from example logs. In support of engineers testing the flight software for NASA's next travel mission to Mars, an analysis framework better known as LOGSCOPE is practiced at the Mars Science Laboratory (MSL) and developed at the Jet Propulsion Laboratory (JPL). In a sense the work signifies an instance of the often pursued marriage between theory and practice in formal methods.

### **Model Inference**

In 2008 an algorithm was presented to derive textitEFSM models of the behavior of software systems called textitGK-tail algorithm[40]. This algorithm can infer behavior models expressing complex interplay between data values and components



interactions. Extended finite state machines (EFSMs) are used as models, where edges are annotated with boolean conditions on data values.

GK-tail processes a set of interactions traces and produces an EFSM that generates all the processed interaction traces. The algorithm work in four steps: merge input-equal traces, generate predicates associated with traces, create an initial EFSM and then merge equivalent state to obtain the final EFSM.

### 3.3 Alternative Approaches

Generating a log is not the only option we have, a common approach applied by developers during the software development is textitdynamic observation and aside from logging there is basically only other method used to analyze program execution of a running system. Developers use tools that enable them to run, pause, resume and trace execution step by step and examine states of the execution instantly online. Ability to examine instantly any variable and object instance at each point of the execution could be the only advantage over a log sistem. However, dynamic observation of running system is not always possible because we are not always supported with the necessary tools.

Dynamic observation is suitable only in some cases and does work for long-term observation. Many times it is not beneficial to monitor running system online and retrieve immediate results, either because we want to defer analysis to avoid performance degradation, or to perform analysis repeatedly.

### 3.4 Web analytics

Web analytics is used to measure, collect, analyze and report website usage data in order to improve the satisfaction of website objectives. Website usage data can include for example the number of unique visitors to the site, total number of page loads, information about how the visitors reached the site, and information about the visitors' browser and operating system. [41, 42]

#### 3.4.1 History and evolution of web analytics

Introduced more than twenty years ago, the internet became commercialized and more and more people had access to it. The early websites were static, created by a single file that could contain text and links.

Working or entertaining, learning or socializing, home or on the road, individually or as a group, Web users are ubiquitously surrounded by an infrastructure of devices, networks and applications. This infrastructure combined with the perpetually growing amount of every imaginable type of information supports the user's intellectual or physical activity. Whether searching, using or creating and disseminating the information, users leave behind a great deal of data revealing their information needs, attitudes, personal and environmental facts. Web designers collect these artifacts in a variety of Web logs for subsequent analysis.

The early websites were static, created by a single file that could contain text and links. As simple as these sites were even they sometimes encountered errors and it didn't take long before someone came up with the idea to take advantage of server logs to obtain information about the number of client requests (also known as hits) made to the web server. Server logs also had additional information about the requests, such as time of request, web browser and operating system used, and a referrer (website sending the request) if a visitor followed a link to the site.

Operational Website management necessitates a way to track and measure visitors' traffic, visitors' behavior and even more importantly how this behavior compares to the expected behavior.

As a growing number of nontechnical people became interested in the data contained in the server logs, it became clear that the data needed to be presented in more understandable form. Soon the first scripts were created that automatically parsed the server log files and created metrics from the gathered data, and thus web analytics was officially born. One of the first server log analysis programs was Analog. [43] Analog is a free software created by Stephen Turner in 1995 and is still widely used.

month:	#reqs:	
-----	-----	
Nov 1995:	119865:	+++++
Dec 1995:	121214:	+++++
Jan 1996:	144960:	+++++

The above display is of a monthly report from early version of Analog, and how basic the first metrics were. With website and web analysis popularity quickly grew, reports also improved appearance. One of the pioneers of commercial web analytics, Webtrends [44], not only made improvements for the standard log file

parsers, but also added tables and graphs to the reports, making server log data even more accessible.

In the late 1990s server log analyzers started to experience challenges in terms of providing accurate data. As the popularity of search engines continued growing, so did the usage of web crawlers. Web crawler, also known as search bot, search engine spider and many other names, is a computer program that browses the Internet automatically or orderly. It is often used by search engines for providing up-to-date data and fast searches, website maintenance for validating Hypertext Markup Language (HTML) code and to ensure that the links work, and by spammers for gathering email addresses. Problem was that the crawler visits to the site were counted by the log analyzers while they were not real visitors.

The next big step in web analytics was the introduction of click analytics. As the name implies, click analytics is a special type of web analytics that focuses on clicks made on the site. Rather than going through piles of charts and tables, decision makers could now see exactly what the users were doing on the site. This also led to increased usage of analytics, since now anyone, not just web analysts, could understand what was happening on the page. Click analytics also made it easier to improve the site based on the visitors' behavior [45].

The evolution of web analytics continues on, as new innovations are made for presenting the complex data gathered from site in newer and easier ways. One of the more recent innovations is a web heat map, which shows the areas where users have clicked most frequently by using colors. The more clicks, the brighter the color. Another recent innovation is a session replay, which can replay what the visitor did on site, including mouse movements, clicks and form entries.

### 3.4.2 Web server logging

The log file that resides in the web server notes the activity of the client who accesses the web server for a web site through the browser.

Web server logs are plain text (ASCII) files and are independent from the server. There are some distinctions between server software, but traditionally there are four types of server logs: transfer log, Agent log, Error log and referrer log. [46]

Server logs, were first used as a means to capture errors encountered on the site and later enhanced to capture additional information. It works in all simplicity

as follows:

- A visitor types the address of the site.
- A request for the page is sent to web server.
- The request is accepted by the web server and an entry about the request is created in the logs. The entry can contain different kinds of information, such as visitor's IP address, browser used et cetera.
- The web server sends the requested web page to the visitor.

Because every web server comes with the means to capture data and create logs, web server logging is probably the easiest way to access data. There are also many log analyzers available for free, so it is easy to start producing basic metrics. Web server logs are usually taken from the server on a set interval, so log analyzers can be configured to produce automatic standard reports. Web server logs are also the only way to gather data about web crawlers, as they do not execute JavaScript tags. It is good to ensure that the web crawlers crawl and index the pages correctly. It is also good to note that data produced with server logs are the property of the website, while most other data-gathering methods sends data to web analytics vendors who capture, analyze and store it. In case of switching web analytics vendors, it is easier to re-analyze old data with new tools if the data is owned by the website.

Data gathered with server logs is mostly technical, however, and not very good for capturing business or marketing information. Cookies are needed for identifying visitors accurately. As server logs capture all the requests, some requests need to be filtered out, such as image or Cascading Style Sheet (CSS) file requests, page errors or web crawler activity in order to get accurate reports about website traffic. ISP caching also causes some traffic data loss because the ISP serves the web page instead of the web server and no data is recorded in the server logs. [45]

### 3.4.3 Tools and web applications

There are two main technical ways of collecting the data. The first method, server log file analysis, reads the logfiles in which the web server records file requests by browsers. The second method, page tagging, uses JavaScript embedded in the site page code to make image requests to a third-party analytics-dedicated server, whenever a page is rendered by a web browser or, if desired, when a mouse click occurs.

Both collect data having a file with some data information but by itself does not help us so much, we must have some tools and web applications to analyse this information. One of this web applications is the famous Google Analytics, but there are many others.

Bellow it is described some of the tools used to make web analytics.

### **Webtrends - analytics 10**

Webtrends is a company, founded un 1993, which develop a software to analyse client/server side web application called Analytics 10. This application uses the log files in which the web server records its all transactions.

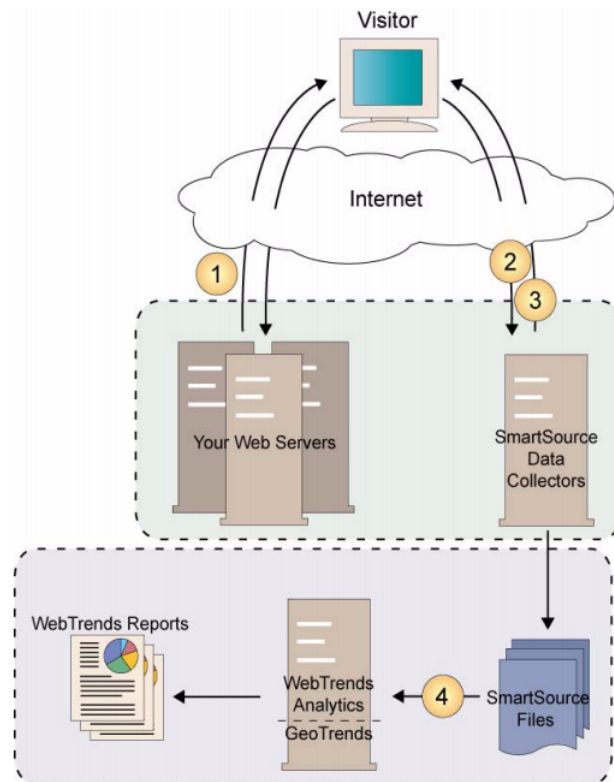
The Webtreds Analytics application process log data and display reports for each configured 'Profile'. Reports are viewable by varios means, most commonly through a web-based graphic user interface or a scheduled emailed export of the report in PDF or CSV format. Collected data can also be viewed via raw log file, REST and ODBC queries, and other solutions which utilize the Webtrends Data Extraction API. The web-based report presentation interface is highly configurable, allowing an administrator to specify what information should be analyzed, where it should be presented, and to whom it should be available for viewing. [47, 48]

The following graphic (figure 3.1) shows how a visitor's interation with one web site is collected, analyzed, and encapsulated in report:

The following steps describe how WebTrends Analytics collects and analyzes data:

1. A visitor requests content from your web site which your web server delivers with the WebTrends JavaScript tag.
2. The WebTrends JavaScript tag causes the visitor's browser to send the SmartSource Data Collectors data about this visitor's web page request.
3. If the visitor is a new visitor, the SmartSource Data Collector server sets a cookie on the visitor's browser and records the page request in SmartSource Data Collector log files.
4. WebTrends Analytics analyzes the SmartSource files and generates WebTrends reports

Figure 3.1: WebTrends Analytics collects and analyzes data



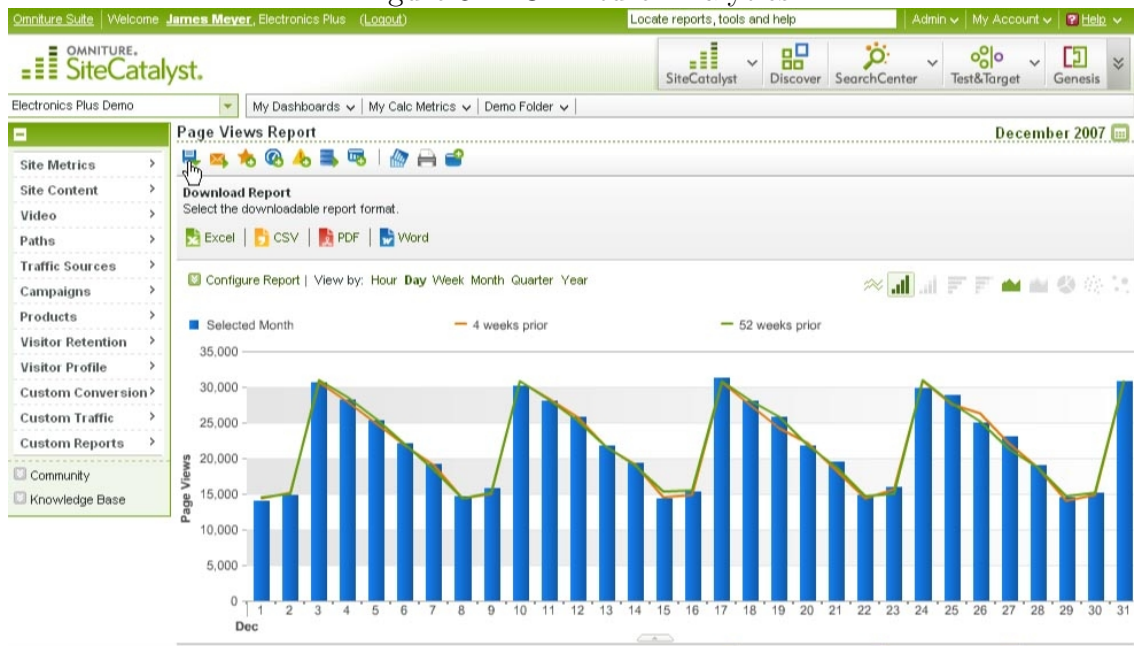
## Omniture

Omniture SiteCatalyst provides marketers with actionable, real-time intelligence about online strategies and marketing initiatives. SiteCatalyst helps marketers quickly identify the most profitable paths through their Web site, determine where visitors are navigating away from their site, and identify critical success metrics for online marketing campaigns. SiteCatalyst is part of the Omniture Online Marketing Suite of applications for online business optimizations.

A wide spectrum of tools and functionality is available for analysts. However, not all these analytic technologies are created equal. Each has its own strengths and limitations and is better suited for certain types of analysis than others. Central to all these tools is the traditional functionality of query, reporting, and analysis. Data quality and data integration tools help to accurately and consistently consolidate data from multiple sources. At the front end, dashboards (figure 3.2) and other visualization tools help users quickly understand analytic results. Lastly, predictive analytics helps to discover hidden patterns and enable what-if analysis.

While Adobe Insight excels in utilizing advanced analysis capabilities to identify

Figure 3.2: Omniture Analytics



patterns and trends in datasets with long-term historical data, it also has powerful capabilities to analyze data in real time. Unlike other systems that provide answers nightly, Adobe Insight is always up to date a set of graphs on an analyst's screen will update continually as new data streams in. This continuous flow of information provides the ability to react quickly and eliminates the need to rerun a time-consuming query again to get up-to-date numbers.

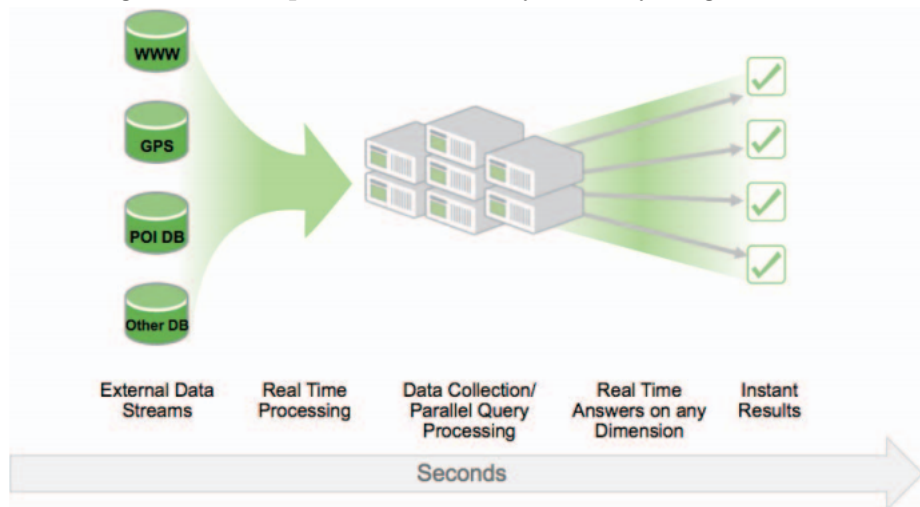
Adobe Insight uses an innovative data storage schema to ensure that every query includes all data that is currently stored in the system. Because the schema is not a relational database and does not use SQL, the results are always correct and are not based on a sampling or aggregation of the data.

Adobe Insight is designed to hold query time relatively constant, regardless of data size. Whether the dataset is a few hundred gigabytes or many terabytes, the analyst receives an instant approximation response for every query and a 100% completed query in minutes, if not seconds. [49, 50]

### IBM coremetrics

[41, 51] Coremetrics is a provider of digital marketing optimization solutions and web analytics and has been one of the key vendors in web analytics industry since the raise of web analytics. The solutions are designed for generating high return for online marketing investments. In 2010, IBM acquired Coremetrics, making it part of IBM's Smarter Commerce initiative.

Figure 3.3: Rapid data discovery for very large datasets



Coremetrics became the integrated web analytics solution for IBM WebSphere Commerce and IBM WebSphere Portal.

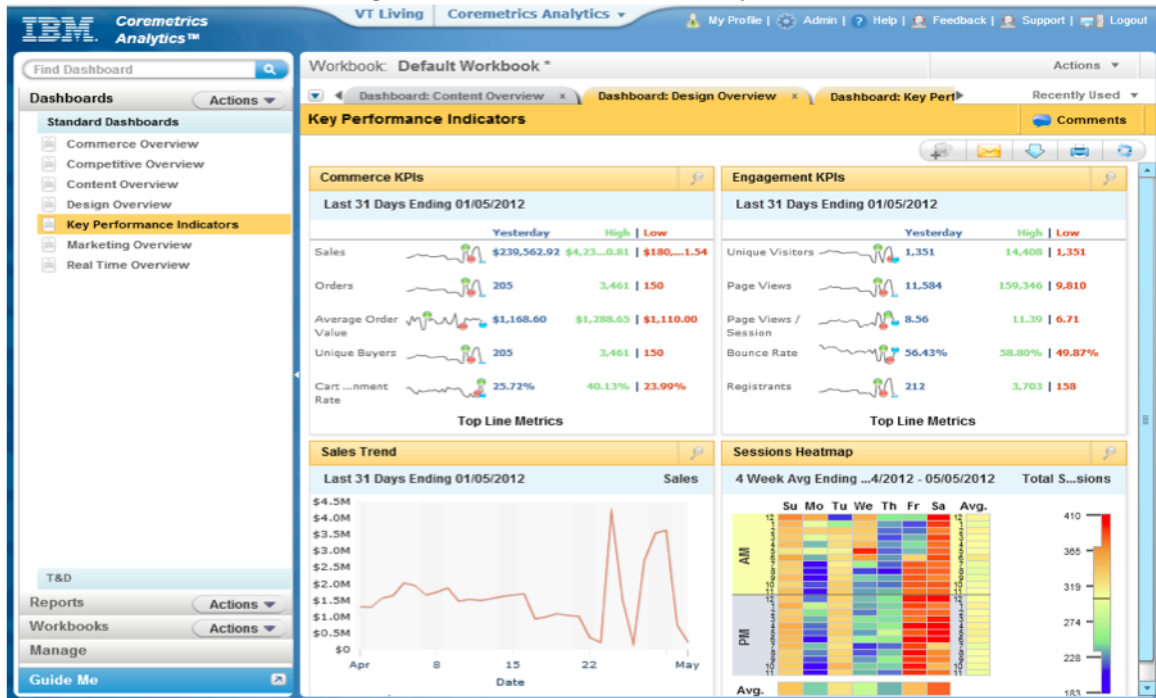
A solution called IBM Coremetrics Digital Marketing Optimization Suite was created by through the fusion of customer profiles, digital marketing execution and testing and web analytics. It offers hundreds of tools and features such as:

- Cross-channel real time analytics and reporting for evaluating marketing effectiveness
- Personalized customer experience based on their lifetime interaction with the business, delivering highly targeted ads and recommendations
- Comparative benchmarks and intelligence about the performance of peers and competitors
- PPC management tool for monitoring PPC campaigns or automated bid management

All the tools and features are available as part of products in the cloud-based Coremetrics application which is accessed with a web browser. The application has an easy to use interface and enables the optimization of web presence and digital marketing channels with flexible and intuitive analytics dashboards, benchmarking and reports. Figure 3.4 has an example view of the Coremetrics Web Analytics application.



Figure 3.4: Coremetrics Analytics



## Urchin

Urchin was Log Analysis software bought by Google in 2005. Urchin software can be run in two different data collection modes; log file analyser or hybrid. As a log file analyser, Urchin processes web server log files in a variety of log file formats. Custom file formats can also be defined. As a hybrid, Urchin combines page tags with log file data to eradicate the limitations of each data collection method in isolation. The result is more accurate web visitor data. Urchin became one of the more popular solutions for website traffic analysis, particularly with ISPs and web hosting providers. This was largely due to its scalability in performance and pricing model.[52, 53]

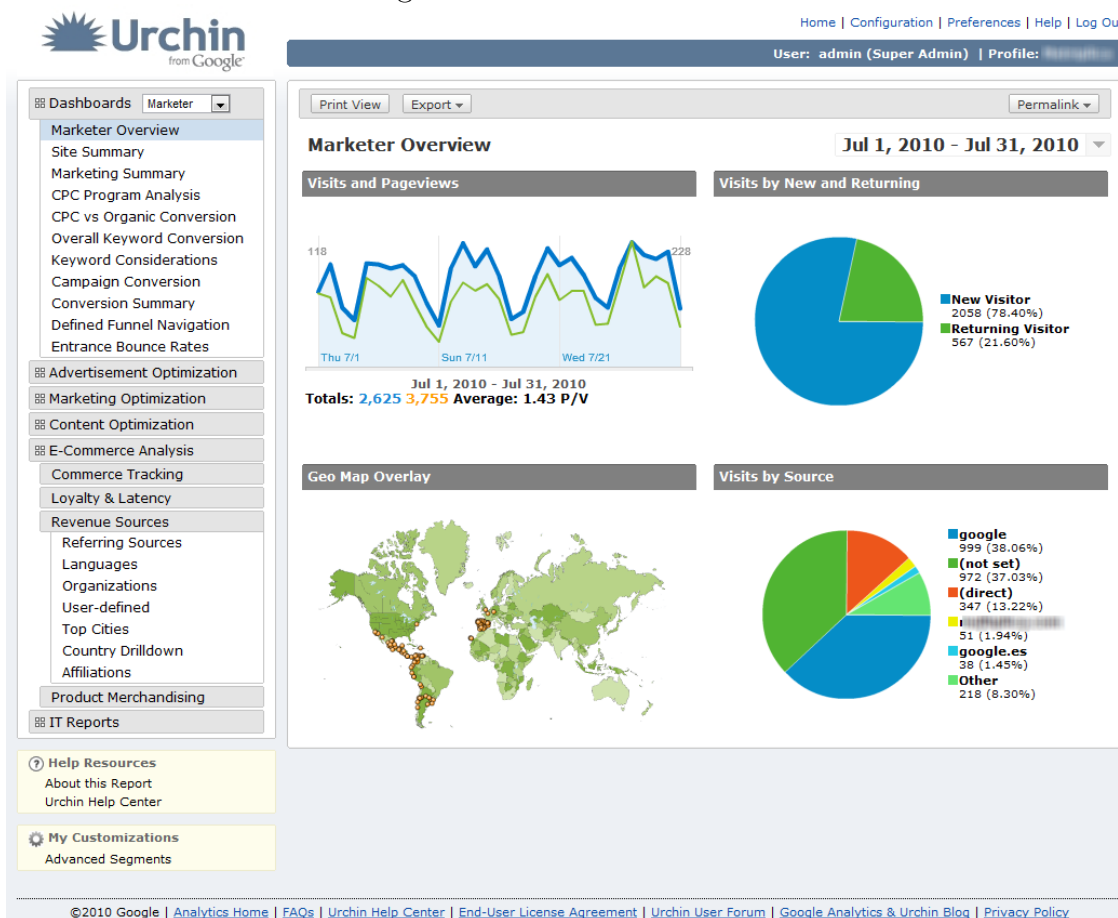
In figure 3.5 we can see the dashboard of Urchin.

Urchin development and support was discontinued by Google on March 2012. They used this software as a base for Google Analytics but have now announced they focus exclusively on Google Analytics.

## Piwik

Piwik is a free and open source web analytics application written by a team of international developers that runs on a PHP/MySQL webserver and one of the best alternative to Urchin. Piwik displays reports regarding the geographic location

Figure 3.5: Urchin dashboard



of visits, the source of visits (i.e. whether they came from a website, directly, or something else), the technical capabilities of visitors (browser, screen size, operating system, etc.), what the visitors did (pages they viewed, actions they took, how they left), the time of visits and more. In addition to these reports, Piwik provides some other features that can help users analyze the data Piwik accumulates, such as:

- **Annotations** the ability to save notes (such as one's analysis of data) and attach them to dates in the past.
- **Transitions** a feature similar to Click path-like features that allows one to see how visitors navigate a website, but different in that it only displays navigation information for one page at a time.
- **Goals** the ability to set goals for actions it is desired for visitors to take (such as visiting a page or buying a product). Piwik will track how many visits result in those actions being taken.
- **E-commerce** the ability to track if and how much people spend on a website.

- **Page Overlay** a feature that displays analytics data overlaid on top of a website.
- **Row Evolution** a feature that displays how metrics change over time within a report.
- **Custom Variables** the ability to attach data, like a user name, to visit data.

Piwik also provides features that are not directly related to analyzing web traffic, including:

- **Privacy Options** the ability to anonymize IP addresses, purge tracking data regularly (but not report data), opt-out support and Do Not Track support. In Germany, 13
- **Scheduled Reports** reports sent regularly by e-mail or text message.
- **Log Importing** a script is also provided that imports data from web server logs.
- **the API** every report is accessible through a web API as well as almost every administrative function. Programs can be created to use this API.
- **the Mobile App** a free mobile app is provided so users can access their analytics data on their phone.

Though not strictly speaking a feature, Piwik also has the characteristic that users are the only people who see their own data. This is a by-product of Piwik being a self-hosted solution.

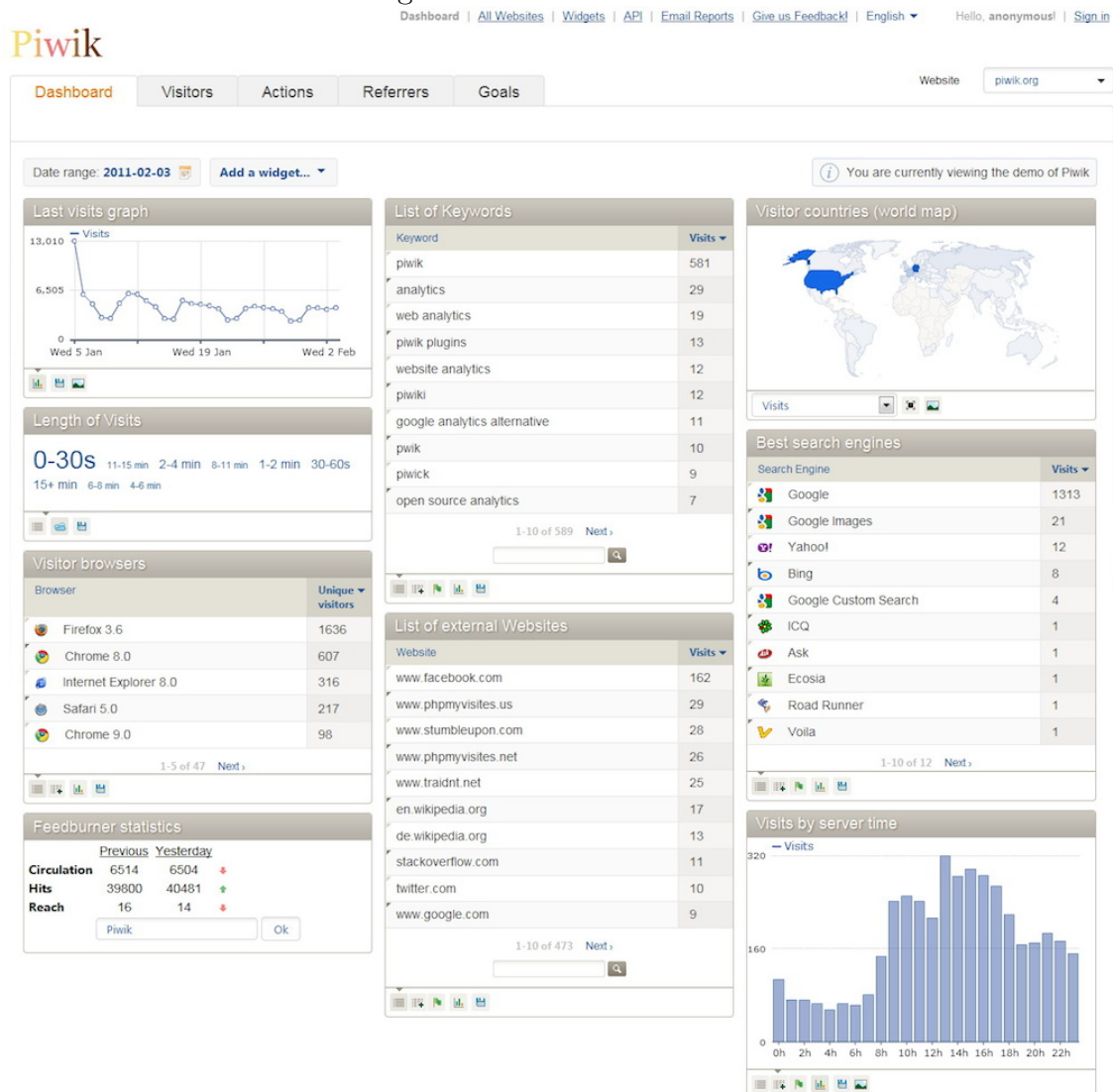
The Piwik log import script was written in Python and is released under the GPL license, for free and have 2 modes:

- **Web Host – web analytics provider user** This mode is ideal for web hosts, where new websites are often added in the access logs, but the Piwik admin does not wish to manually create each website. The script will automatically detect the Piwik website ID to track based on the URL being parsed: it will look for any Piwik website registered with a URL or “Alias URL” set to this page view host. If a website with the hostname doesn’t exist, a new website is automatically created for this URL. A summary is then emailed to the Piwik Super User so he/she knows which websites are automatically created by the log import script and can create users or assign permissions to view these new websites.

- **Simple log import for one or a few websites only** This mode is ideal if you import only a small number of websites or if you wish to control exactly in which websites requests are to be tracked. When a line contains a URL to an unknown Piwik website, Piwik will ignore all these page views and will report, at the end of the script execution, the list of hostnames that were not matched to any website in Piwik. If these unknown URLs turn out to be legitimate page views, you can either create a new website manually, or add an Alias URL to an existing website, so the page URLs are directly tracked in this website the next time you import similar logs. [54]

We can see the Piwik dashboard on Figure 3.6.

Figure 3.6: Piwik dashboard





# Chapter 4

## FITTEST log Format

We can think about logs as a plain text file, where every line represents either an abstraction of an event in the system, or the system's internal state at a concrete moment, or both.

Abstractly we can view an execution of a program as a sequence of events; each one may change the state of the program. Some of these events may be interesting to be logged. Furthermore, we can define the concept of events at different levels of granularity. E.g. we can consider each machine instruction as an event; this would be very low level. On the other hand, we can choose function calls to be our events. It also makes sense to log events of different granularity levels.

The highest level of granularity is at the program's interactions with users (or other external entities that actively use the application), for example, when an user fills in a text field, or when he clicks on a button. We call these interactions "application events" or "high level events". Logging these events help us understand how a user uses the application and how he affects it, e.g. when an error or something atypical is observed.

On the other hand, the lower level of granularity is when some functions are called or exited when some exceptions are thrown, or when some branches are passed. Logging these lower level events give us deeper insight on what happens during an execution, but it is more expensive and hence should be done selectively. We call the logging of lower level events deep logging.

## 4.1 Syntax

The FITTEST logs use the UTF-8 encoding rather than binary and are stored in files with the .log extension.

A FITTEST log is just a sequence of log entries where each entry is organized as a section, which can recursively be made of sub-sections. The lowest level section is called paragraph, which consists of sentences, which are strings of text.

Sections are tagged, which can then be used to associate some semantics to them. We can view the syntax below.

```

<log> ::= (<section> <whites>)+
<section> ::= <section start> <whites> (<section part><whites>)* <
end marker>
<section start> ::= %<S <whites> <time stamp>? <whites> “<tag>”
<end marker> ::= %>
<whites> ::= <white>*
<section part> ::= <paragraph> | <section>
<paragraph> ::= %<p <whites> (<sentence><whites>)* <end marker>
<sentence> ::= %<{ <sentence content> }%>
<time stamp> ::= <UTC offset> : <UTC time>
<UTC offset> ::= (+/-)? <offset in minutes>

```

Additional constraints:

- <sentence content> is any sequence of character, but it should not contain the combination, which is used to identify the sentence’s end.
- Time stamp is written as a pair “o : t” where “t” is UTC time, which is location independent, and “o” is the offset of the local time with respect to the UTC time. With this offset we can infer what the local time of “t” is. The UTC-time “t” encoded as a single integer, which expresses the number of milli-seconds elapsing since midnight 1-st January 1970 and the actual UTC time when “t” is measured. The offset “o” is also an integer, expressing the time difference

between the location on which “t” is measured and the UTC time, expressed in minutes.

## 4.2 Base Tags

This section defines a set of base tags used by the FITTEST format. A subset of these tags is used to describe events. FITTEST format also allow objects’ state to be logged, so the other subset of tags are related to object serialization.

the following meta notation has been used for describing the tags’ syntax:

1. Sen(x) means that “x” will be formatted as a sentence, with “x” as the content.
2. Par(S<sub>1</sub>...S<sub>n</sub>) means that this will be formatted as a paragraph with each s<sub>i</sub> forming a sentence of the paragraph, appearing in the order as specified.
3. Sec(T,S<sub>1</sub>...S<sub>n</sub>) means that this will be formatted as a section with tag T, with the S<sub>i</sub>’s concatenated to form the body of i-th section. This section has no timestamp.
4. Sectimed(T,S<sub>1</sub>...S<sub>n</sub>) is as above, except that the section can be timestamped.

### 4.2.1 Object Format

In this section is defined how objects should be printed/serialized to the log. The concept of ‘object’ broadly represents a structure of some data. It can be a real object in the target program, or a fake object that is used to abstractly represent a complicated object.

Because a FITTEST log is defined as a sequence of events, objects are never logged on their own, so they must be part of events. There is two kinds of objects: simple and nested. A simple object or ‘value’ only has a single paragraph with a single sentence.

```

<object> ::= <simple object> | <nested object>
<simple object> ::= Par(Sen(<simple value>))
<simple value> ::= <undefined> | <>null> | <integer>
| <numeric> | <boolean> | <string> | <unseriazable>
<undefined> ::= undefined:void

```



```

<null> ::= null:null
<integer> ::= <int value>:int
<numeric> ::= <numeric value>:Number
<boolean> ::= <Boolean value>:Boolean
<string> ::= <string value>:String
<unseriazable> ::= ??:<class name>

```

Examples of simple objects

```

%<P %<{ undefined:void }%> %>
%<P %<{ 789:int }%> %>
%<P %<{ 3.421:Number }%> %>
%<P %<{ true:Boolean }%> %>
%<P %<{ 'hello world!':String }%> %>

```

A nested object has multiple fields, some of these may contain sub-objects. These fields are grouped in one or more paragraphs, which can be exploited for compression, e.g. the logger may group the fields such those that tend to vary together are put in the same paragraph.

```

<nested object> ::= Sec(<object tag>,<object body>)
<object tag> ::= 0:<class name>
<object body> ::= <object part>*
<object part> ::= Par(<simple field>1 ... <simple field>n
| Par(<simple field>1 ... <simple field>n <subobject marker>))
<nested object>
<simple field> ::= Sen(<field name> = <simple value>)
                | Sen(<field name> = ^<iref>)
<subobject market> ::= <field name> =>

```

In any case, the logger is free to decide how to arrange the fields in paragraphs, but they must be packed in paragraphs. There is however one restriction imposed by the syntax. Consider as an example this class:

```

Class Person {
    var name : String;
    var age : int;
    var spouse: Person;

```

```

        var height: int;
    }

```

The spouse field of a person contains another person as a sub-object. This cannot be packed in a paragraph, and has to be packed as a section instead. However, the field name (spouse) should be put in the paragraph that precedes it. For example, here is how a person can be serialized:

```

<S ‘0:Person’
    %<P    %<{ I=0:ID }%>
            %<{ name=’Sponge Bob’:String }%>
            %<{ age=4:int }%>
            %<{ spouse=> }%>
    %>
    <S ‘0:Person’
        . . . // here is the serialization of the spouse
    %>
%>

```

## 4.2.2 Event Format

As previously mentioned, there are two types of events: the high level and low level.

### High level events

A high level event “e” is described by an ‘event object’ and a state object. The first describes what kind of event “e” is (e.g. a click on a button, or an update to a text-field), and the second abstractly describes the state of the target application ‘when’ the event occurs. We do not impose here what the exact temporal relation between the sampled state and the event should be; this depends on the implementation of the logger. Typically, the state is sampled after the event occurs, and before the next event occurs.

```

<high level event> ::= Sectimed (E, <event object> <state object>)
    <event object> ::= <nested object>
    <state object> ::= <object>

```

### Example of High level Event

```

%<S -120:1354468506 "E"
  %<S "0:HighLevelEvent"
    %<P %<{I=0:ID}%>
      %<{ targetID="ButtonHome":String}%>
      %<{ type="homepageclick":String}%>
      %<{ args=> }%>
%>

  %<S "0:Array"
    %<P %<{I=1:ID }%>
    %<{elem=1:int}%>
    %>
  %>
%>
  %<S "0:State"
    %<P %<{I=0:ID}%>
      %<{ php_page="index.php":String}%>
      %<{ user="NoUser":String}%>
      %<{ name ="NoName":String}%>
      %<{ type=:NULL}%>
    %>
  %>
%>

```

The event must have the following fields and they are fixed:

- Type:String -> Specifies what kind of event it is. E.g “homepageclick” means the event was a click on some homepage button.
- TargetID:String -> String that identifies the display object that is targeted by the event. E.g. if the event was a click on a button “b”, this this would be the ID-name of “b”. This ID is usually unique.
- Args:Array -> Array containing the arguments passed to the event. Events like clicking on a button have no argument. On the other hand, an update to a text-field is an event that takes one argument, namely the new value in the text-field.

### Low level events

FITTEST provides events that can be used to log the entrance and exit of functions, and visits to instructions blocks inside functions. The events of type FE (function entry) and FX (function exit) are used to log at the entrance and exits of a function. The syntax is below:

```

<function entry> ::= Sectimed (<function entry tag>, <target object>
<args>)
<function entry tag> ::= FE:<function name>

<function exit> ::= Sectimed (<function exit tag>, <target object>
<return object>)
<function exit tag> ::= FX:<function name>

<function name> ::= <function entry tag>:<target object>
<return object> ::= <object>
<target object> ::= <object>
<args> ::= Sec(args <object>*)

```

The program of a method can be divided into 'blocks'. A block is a maximal consecutive segment of instructions that does not contain a jump nor targeted by a jump. The events below can be dispatched to log when a block is visited.[2]

```

<visit block event> ::= Sectimed (<visit block tag>, )
< visit block tag > ::= B:<block id> : <function name>

```

Low level event example of a pair of FE and FX events describing move(2,3) where move is a function/method of the class Point. In the example, the arguments are simple values, but in principle they can also be objects. In the middle we see block events that corresponds to the instructions-blocks the execution of the call move(2, 3) internally pass. The block events are not mandatory, they are showed here just as an example. [2]

```

%<S -120:1347473178132 "FE:move:Point"
  %<S "0:Point"
    %<P %{I = 0:ID }%>
      %<{ x=10:int }%>

```

```

                                %<{ y=10:int }%> %>
%>
%<S ‘args’
                                %<{ 2:int }%>
                                %<{ 3:int }%>
%>
%<S -120:1347473178141 ‘B:543:move:Point’ %>
%<S -120:1347473178141 ‘B:612:move:Point’ %>
%<S -120:1347473178142 ‘B:632:move:Point’ %>
%<S -120:1347473178142 ‘B:640:move:Point’ %>
%< -120:1347473178143 ‘FX:move:Point’
    %<S ‘O:Point’
                                %<P %<{ I=0:ID }%>
                                %<{ x=12:int }%>
                                %<{ y=13:int }%> %>
%>
%<P %<{ undefined:void } %> %>
%>

```

### 4.3 FITTEST log file to XML log file

The FITTEST log format can be also converted into a XML log file, however the FITTEST log format is more compact. There is an accompanying line program called haslog that can convert FITTEST logs to an XML format, so for analysis we may prefer XML because of the availability of various XML-based tools. Haslog can also “compress” logs and it may or may not result in smaller logs. The compression exploits values in the FITTEST logs that are grouping in paragraphs. The effectiveness of this depends on how smart the used logger did the grouping. To convert a FITTEST format file to XML file we can follow the commands below.

- `haslog -c FIT.log` (make compression of FITTEST log file)
- `haslog -x FIT.lox` (using compressed file and convert to a XML file)

In this case it will produce the file FIT.xml This could be worthwhile so that people can use XML tools to inspect the logs, e.g. they can write XPath/XQuery queries on the logs.[2]

## 4.4 XML log file

This section describes the structure of the resulting XML file. The elements of this XML has been described notations like this:

```
element E = attrib optional t
content <event object> <state object>
```

This defines an element with E as the tag, and it has an optional attribute called t, and has two sub- elements: an event object and a state obect. An incomplete example of an instance of this element is here:

```
<E t =" -120:1312787896474" >
  ... // event object
  ... // state object
</E>
```

The top-level element is body, which consists of entries as sub-elements. An entry can be a high level event, or a low level event.

```
element body = content <entry>*
<entry> = <high level event> | <low level event>
<high level event> = E
```

Here is the structure of a high level event:

```
element E = —high level event
attrib optional t —time stamp
attrib content <event object> <state object>
```

```
<event object> = O
```

```
<state object> = <object>
```

```
<object> = O | V
```

```
element O = —nested object
```

```
attrib type —the object's type
```

```
content fd* —fields
```

```
element fd = —the object's fields
```

**attrib** n —field name

**content** (O | V) —field's content

**element** V = —simple object

**attrib** v —the object's value

**content** ty —the object's type

An example of a log in the XML format:

```
<E t =" -120:1312787896474" >
  <O ty="eu.fittest.actionscript.automation::RecordEvent">
    <fd n="I">
      <V v="0" ty="ID" />
    </fd>
    <fd n="targetID">
      <V v="&quot ; ButtonBar0&quot ;" ty="String " />
    </fd>
    <fd n="type">
      <V v="&quot;itemclick&quot;" ty="String" />
    </fd>
    <fd n="args">
      <O ty="Array">
        <fd n="I">
          <V v="1" ty="ID" />
        </fd>
        <fd n="elem">
          <V v="1" ty="int" />
        </fd>
      </O>
    </fd>
  </O>
  <O ty="AppAbstractState">
    <fd n="I">
      <V v="0" ty="ID" />
    </fd>
    <fd n="numOfSelectedItems">
      <V v="18" ty="int" />
    </fd>
  </O>
</E>
```

```

    </fd>
    <fd n="numInShopCart">
      <V v="0" ty="int" />
    </fd>
    <fd n="cartCurrency">
      <V v="&quot;$&quot;" ty="String" />
    </fd>
    <fd n="cartTotal">
      <V v="&quot;$0.00&quot;" ty="String" />
    </fd>
  </O>
</E>

```

Here is the structure of a low level event:

```
<low level event> = FCE | FCX | FE | FX | B | BEH | BLE | BLX
```

**element** FCE = —function call entry event

**attrib** f —caller function name

**attrib** ce —caller name

**content** <target object> args

<target object> = <object>\*

**element** args = **content** <object>\*

**element** FCX = —function call exit event

**attrib** f —caller function name

**attrib** ce —caller name

**content** <target object> <return object> <exception object>

**element** FE = —function entry event

**attrib** f —function name

**content** <target object> args

**element** FX = —function exit event

**attrib** f —function name



**content** <target object> <return object>

**element** B = —visit block event

**attrib** f —function name

**attrib** i —block ID

**element** BEH = —handling exception event

**attrib** f —function name

**attrib** i —block ID

**content** <exception object>

**element** BLE = —enter loop event

**attrib** f —function name

**attrib** i —block ID

**element** BLE = —exit loop event

**attrib** f —function name

**attrib** i —block ID

**attrib** i —block ID

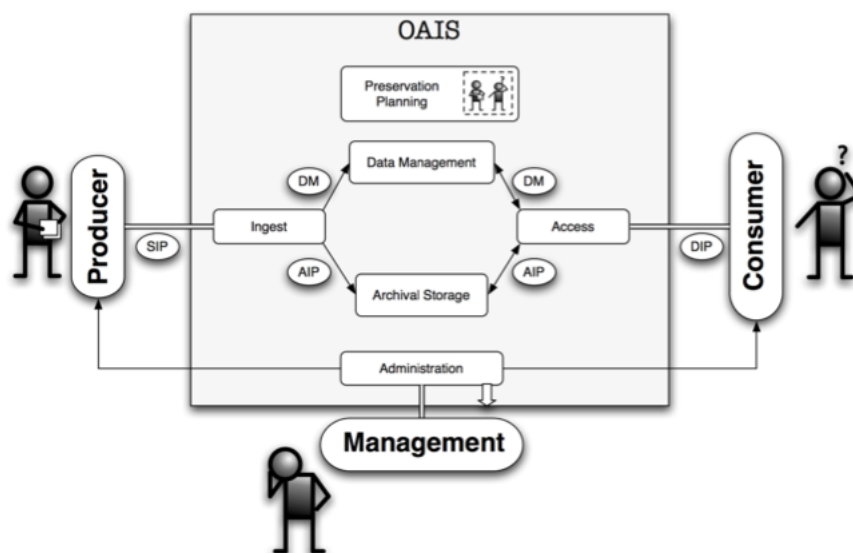
FITTEST is an incessant testing method that has been chosen to handle the augmented dynamics in forthcoming internet applications. These main components will be determined by the feedback gathered at run-time to adjust current models and test-cases, and to conclude new ones. A crucial source for this feedback, are log files. Several applications service logging in order to provide tracing information about their performances. Overall, generated logs offer a large amount of information that can be valuable in numerous ways.

# Chapter 5

## Test Example

To develop this project, it would be necessary to use a test application (in this case, a webpage) to generate logs. The webpage used was developed as part of an old master project, for the course of “Structured Processing of Documents”, and it’s an online repository of text reports. The main goal of this old project involved the development of a web application, to be used by a teacher, and to act as a digital repository for the reports of all the students. The objective was to safely store this documents and at the same time to allow online access to them. The repository was implemented following the guidelines of the OAIS model (Figure 5.1)

Figure 5.1: OAIS model



The OAIS model consists of 3 mega processes:

- **Ingestion** process that deals with the incorporation of new information into the system;

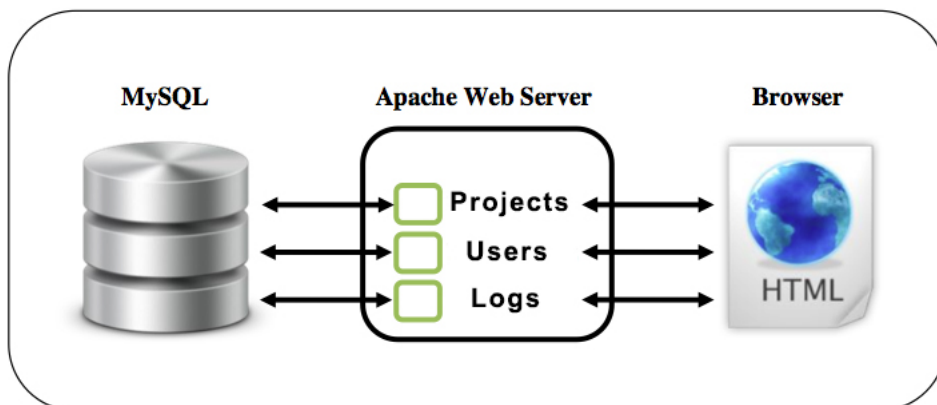
- **Administration** process that deals with the management of information within the system;
- **Dissemination** process that deals with the exploitation and distribution of the information stored.

The repository contains three types of users: Producer, Consumer and Administrator.

- **Producer** User who provide information to the system. It can be done so be assisted through a Web interface or in Batch by sending SIP ("Submission Information Package") which is a "package" of information with previously negotiated and specified structure between producers and repository;
- **Consumer** The consumer is the end user of the system that interacts with the repository through a specific API that allow to perform a series of operations: Download a set of information packaged in a DIP ("Dissemination Information Package"), explore/visit a particular virtual exhibition through the navigation on a website created for this purpose, etc. Each operation accessible to this user correspond the implementation of a disseminator. There is no limit to the number of disseminators.
- **Administrator** The Administrator conducts a series of internal operations: creation and control of users, creating and storing ontologies for classification and definition of virtual exhibitions, etc.

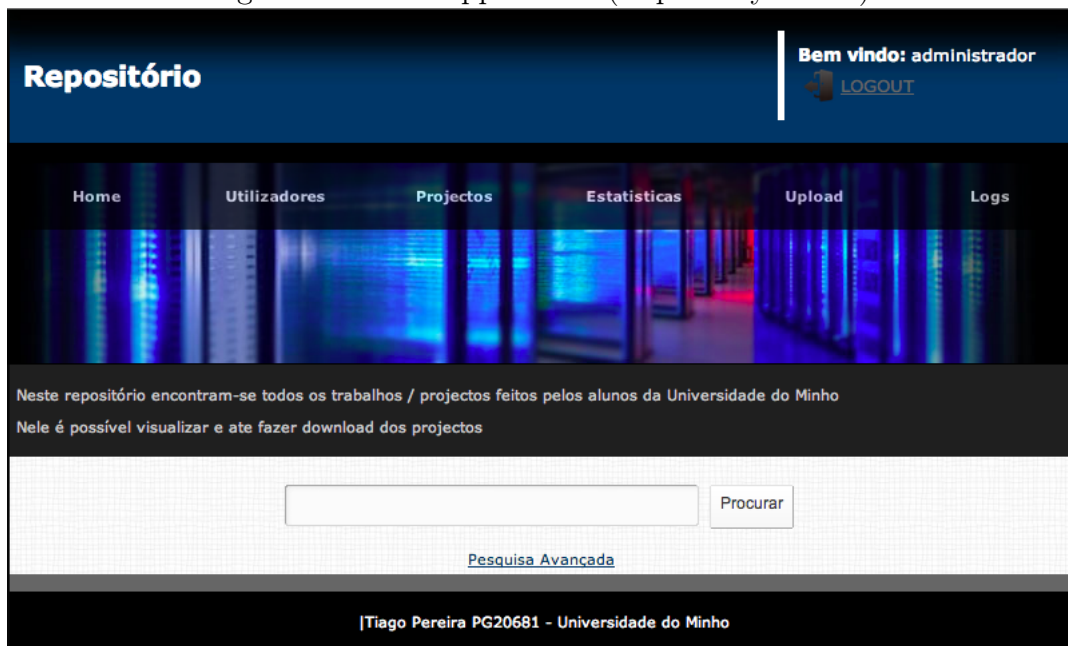
The architecture of the repository can be viewed in Figure 5.2

Figure 5.2: Test Application (Repository architecture).



The homepage of the repository can be viewed on Figure 5.3. This application

Figure 5.3: Test Application (Repository online).



became a “digital portal” with free and organized access to the reports by the students, promoting a more eco-friendly solution to the old paper reports that would need to be delivered in each assignment.

## 5.1 Repository Layout

The entire repository layout has been developed using HTML and CSS. It was also used some JavaScript plugins for some parts of the repository, namely, tables, buttons, datapicks, forms, alert messages and search field.

For the tables it was used a plugin called tablesorter, that can transform a simple table into something more dynamic, enabling sorting columns by crescent or descending order and applying a design (zebra effect)

Another plugin called JQuery Ui was used on buttons, datapicks and alert messages, and on forms it was used the jqtransform plugin which automatically applies a css on buttons, textboxes, text areas, radio buttons, comboboxes and checkboxes. Lastly it was used a JQuerySearch plugin which does some animations to the webpage, making the search field expand when clicked on.

All access attempts to restrictive pages, administrators or producers has been blocked

as shown on figure 5.4.

Figure 5.4: Test Application (Access Denied).

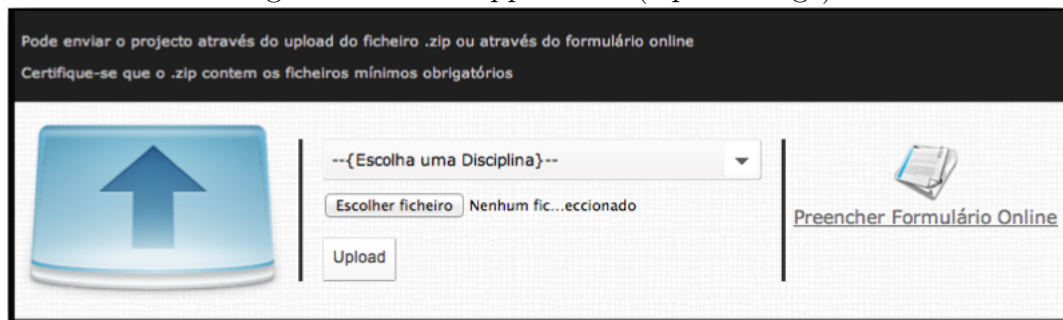


## 5.2 Projects

There are two major parts regarding projects, one regarding project deposits (SIP) and the whole process of intake, and the second part regarding listing, maintenance and design visualization (DIP).

The intake process in the repository can only be done by an administrator or by a producer, and can only be done in two ways, by sending a zip file, as shown in figure 5.5 or by filling out an online form as shown in figure 5.6.

Figure 5.5: Test Application (Upload Page).



In any chosen method to send the project, is necessary to choose a discipline to which it is intended.

In the case where we submit a project in the .zip format, it is analyzed rigorously, checking it is a zip file, if it contains a pr.xml file and attachments (if any), if the pr.xml structure coincides with an xml schema and the final date is not greater than the current date. If one of these tests fails, an error message is displayed redirecting us automatically to the upload page, if it is successful, it displays a

Figure 5.6: Test Application (Upload Form).

The form is titled "Projecto:" and contains the following fields:

- Disciplina:\* --{Escolha uma Disciplina}-- (dropdown menu)
- keyname:\* (text input)
- Titulo:\* (text input)
- Data Inicio:\* (text input)
- Sub.Titulo: (text input)
- Data Fim:\* (text input)

The "Equipa de Trabalho:" section contains:

- 1 (dropdown menu) and Seleccionar (button)
- Nome:\* (text input)
- Numero:\* (text input)
- Email:\* (text input)
- Url: (text input)

The "Supervisores:" section contains:

- 1 (dropdown menu) and Seleccionar (button)
- Nome:\* (text input)
- Email:\* (text input)
- URL: (text input)
- Affil: (text input)

The "Sumário:" section is a large empty text area.

The "Ficheiros Anexo:" section contains:

- 1 (dropdown menu) and Seleccionar (button)

At the bottom right of the form is an "Enviar" button.

message indicating that the project was sent.

In the case of the submission of a form, it will also be checked for errors, like when one of the required fields was not filled. For the date fields, users have the help of a datapick, to prevent incorrectly inserted dates.

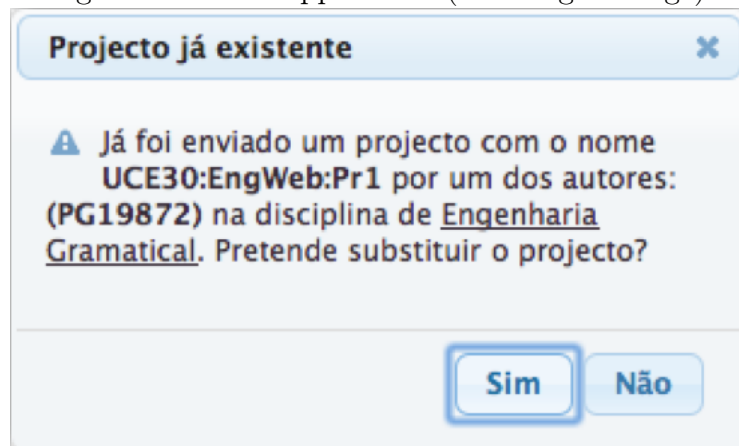
There is a common verification, which verifies if the submitted project already exists, or if it has already been submitted by another user who makes part of the project. If the project already exists, a message will appear (Figure 5.7) and the user will be prompted to choose whether to overwrite or keep the existing project.

All submitted projects are stored on the server in a folder called 'upload'.

A folder structure was created to organize the projects, preventing users overwriting projects with the same name.

Each project will be stored within a folder labeled with the project name (eg UCE30-

Figure 5.7: Test Application (Warning message).






EngWeb-Pr1) and this within a folder with the student number which carried out the work (eg PG19872), and lastly it will be stored in another folder with the name of the discipline to which the project relates. An example of a project path would be: "upload / Engineering Gramatical/PG19872/UCE30-EngWeb-Pr1".

Displaying a project and viewing a detailed listing can be done by any user, even the ones that are not registered. The listing can be viewed in figure 5.8

Figure 5.8: Test Application (Project List).

The screenshot shows a web interface titled "Download Projects" with a search bar labeled "Procurar Projecto". Below the search bar is a table with the following columns: Keyname, Título, Data Submissão, Equipa de Trabalho, Supervisores, and three action buttons (green arrow, blue arrow, red X). The table contains four rows of project data.

Keyname	Título	Data Submissão	Equipa de Trabalho	Supervisores			
<a href="#">UCE30:EngWeb:Pr1</a>	Um título Qualquer	2012-01-22 14:26:01	• marco costa • Tatiana Pinto	• José Carlos Ramalho • Outro Prof			
<a href="#">UCE30:EngWeb:Pr2</a>	Gerar texto aleatório	2012-01-22 14:02:19	• marco costa • Tatiana Pinto	• José Carlos Ramalho • Outro Prof			
<a href="#">UCE30:EngWeb:Pr4</a>	tis unde omnis iste natus erro	2012-01-22 00:51:14	• marco costa • Tatiana Pinto	• José Carlos Ramalho • Outro Prof			
<p>▶ <a href="#">LINKS ON WEB.txt</a></p> <p>▶ <a href="#">pr.xml</a></p>							
<a href="#">UCE30:EngWeb:Pr3</a>	Modelo de Dados para o Repositório	2012-01-22 00:50:58	• maria manuela • Tiago Pereira • Manel das cenouras	• José Carlos Ramalho • Outro Prof			

On this page the user can see some information regarding the project, his attachments via the button , download the project by clicking . Clicking delete on the button  is restricted, only administrators are allowed to do this. The user can also search by the project's keyname or title. This feature is also present for users and logins.


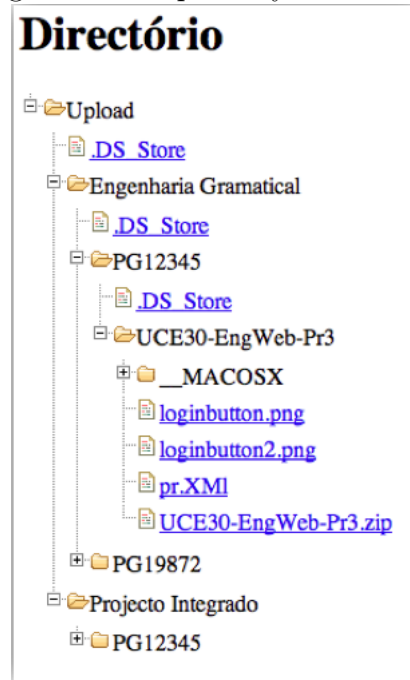


The administrators can download all existing projects repository through the icon . The downloaded .zip contains all projects (properly organized) and an html file to view all existing projects in the upload folder as shown in figure 5.9

Figure 5.9: Repository HTML tree



A tree was created using the javascript plugin JQuerytree to facilitate the iteration and the visualization of the process.

In the same list, clicking on a keyname, will be shown all the details regarding it, as shown in figure 5.10

On this page the user can read all of the information regarding the project and can navigate through them by using the arrows   without needing to return to the listing page, allowing also to download the project at any given time.

Both the teamwork and supervisors, only the name will be shown, but once we click on a name, a little popup will appear, containing all of the detailed information concerning the person who we clicked on as shown in figure 5.11.

To create the popup a javascript plugin called fancybox was used.

## 5.3 Users

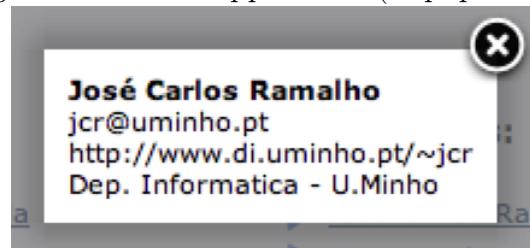
This section pertains to the entire user management by administrator or the registration of a new repository consumer. The administrator can see all registered users in the repository through a list as shown in figure 5.12. In this page the ad-



Figure 5.10: Test Application (Project Detail).



Figure 5.11: Test Application (Popup Detail).







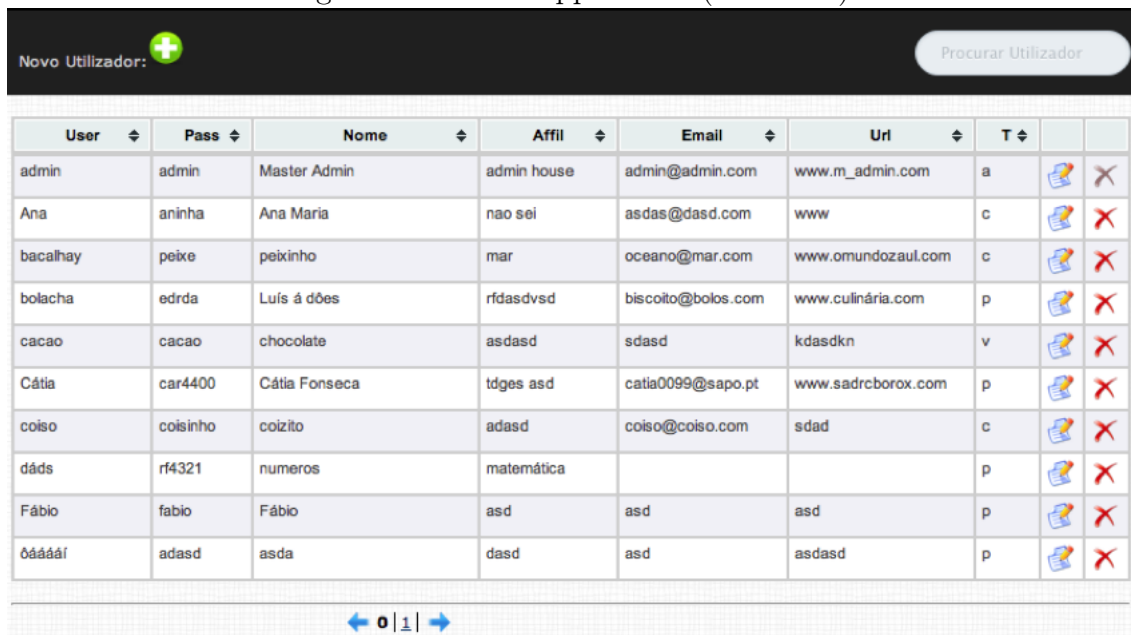
administrator can see all registered users information and can add a new ones , edit  or remove  the existing ones. The administrator who is logged cannot remove himself, that distinction is made a button more faded . New users can be added by register button or in the administration page. The form can be visible in Figure 5.13. The only difference between administrator or an unregistered user is that the administrator can assign a type of user (Administrator, producer or consumer). When it's entered or edited a user, some checks are made such as: if the username already exists, or if the required fields are properly filled. If the verification fails, an error message will be displayed (Figure 5.14). A message as shown in Figure 5.15

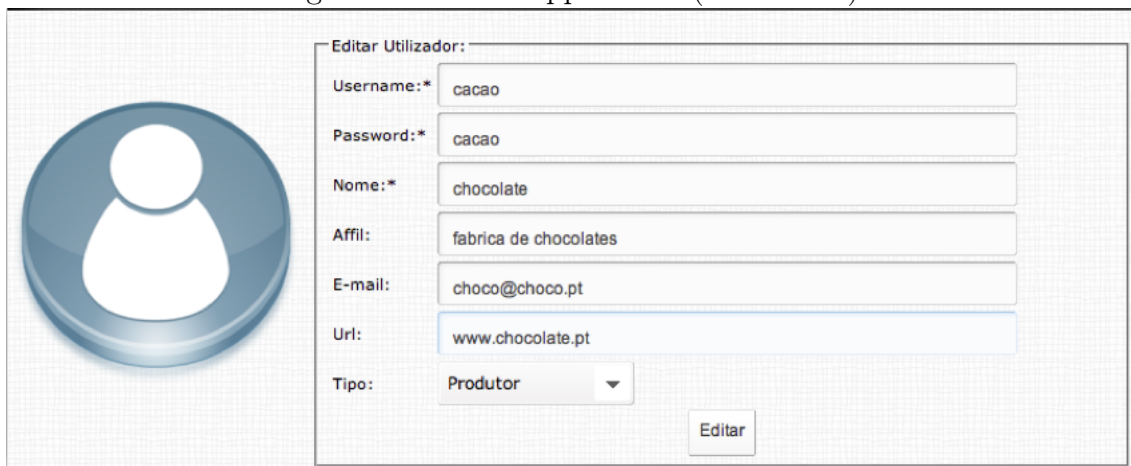
Figure 5.12: Test Application (User List).



User	Pass	Nome	Affil	Email	Url	T		
admin	admin	Master Admin	admin house	admin@admin.com	www.m_admin.com	a		
Ana	aninha	Ana Maria	nao sei	asdas@dasd.com	www	c		
bacalhay	peixe	peixinho	mar	oceano@mar.com	www.omundozaul.com	c		
bolacha	edrda	Luis á dôes	rfdasdvsd	biscoito@bolos.com	www.culinária.com	p		
cacao	cacao	chocolate	asdasd	sdasd	kdasdkn	v		
Cátia	car4400	Cátia Fonseca	tdges asd	catia0099@sapo.pt	www.sadrcborox.com	p		
coiso	coisinho	coizito	adasd	coiso@coiso.com	sdad	c		
dáds	rf4321	numeros	matemática			p		
Fábio	fabio	Fábio	asd	asd	asd	p		
óááááí	adasd	asda	dasd	asd	asdasd	p		

← 0 | 1 | →

Figure 5.13: Test Application (User Form).



Editar Utilizador:

Username:\* cacao

Password:\* cacao

Nome:\* chocolate

Affil: fabrica de chocolates

E-mail: choco@choco.pt

Url: www.chocolate.pt

Tipo: Produtor

Editar

will appear if the user has been inserted or edit successfully.

Figure 5.14: Test Application (Error Message).

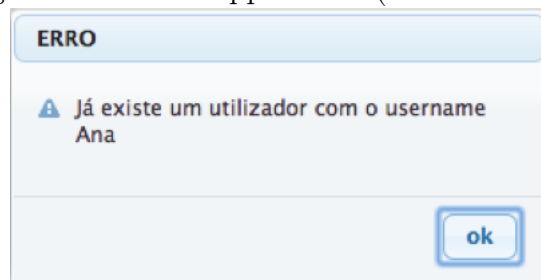
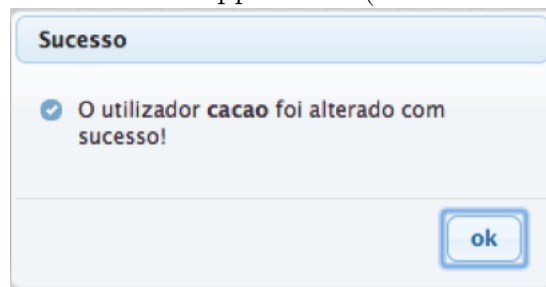


Figure 5.15: Test Application (Success Message).



## 5.4 Statistics

Some statistics regarding the repository usage can be viewed by the system administrator. Those statistics are divided in three parts:

- user statistics
- project statistics
- discipline statistics

In the user statistics it can be seen for each user, the number of deposits made (with a link to view the deposited projects), the number of times he has accessed the repository, the number of views and also the number of downloads. “It’s possible to know the total number of deposits, those existing as those that have already been deleted and the total number of accesses separating the access of registered users, unregistered and existing.

In project’s statistics it is possible to know for each project the number of times it has been viewed and how many times they’ve been downloaded.

The administrator can list the top 10 most viewed projects, or the top 10 most downloaded ones.

In subject statistics, it is shown for each subject, the total number of projects relating to the same and the number of views.

An example of a statistics page can be viewed in figure 5.16. The majority of the statistics are being calculated by the xml’s log, applying xpath filters. Only the numbers of deposits(in the case of users) and the number of views and downloads of the project are calculated with access to the database. Still in the statistics page, the administrator can generate a graph, with a similar appearance to figure 5.17 A javascript plugin called amcharts has been used to generate graphs.

Figure 5.16: Test Application (Statistics page).

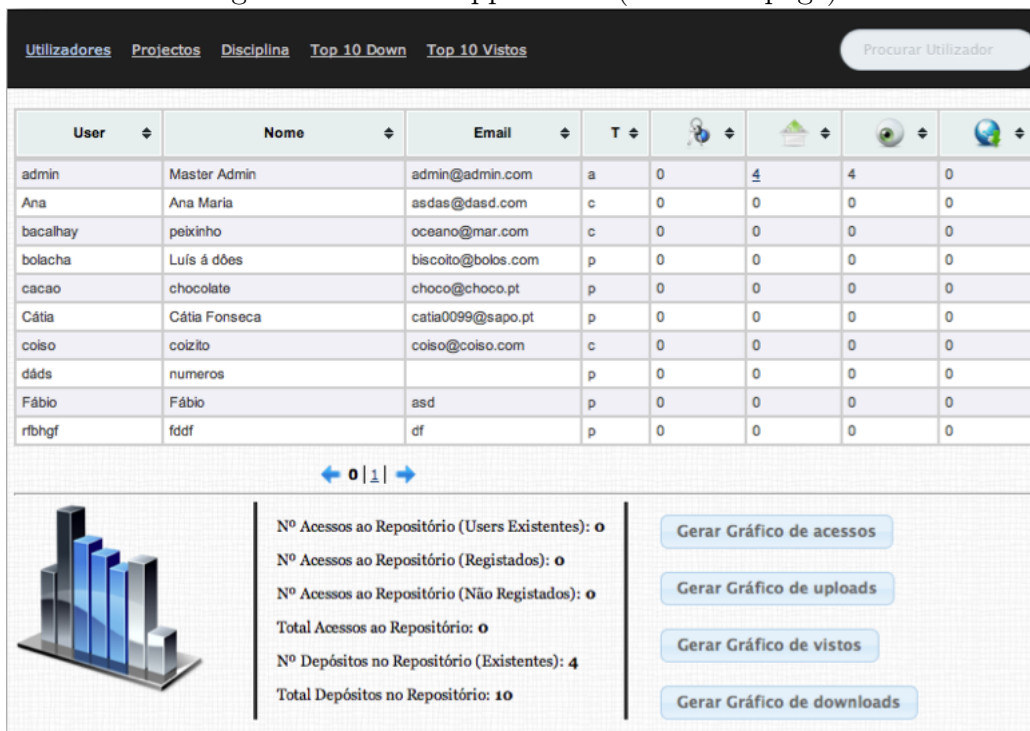
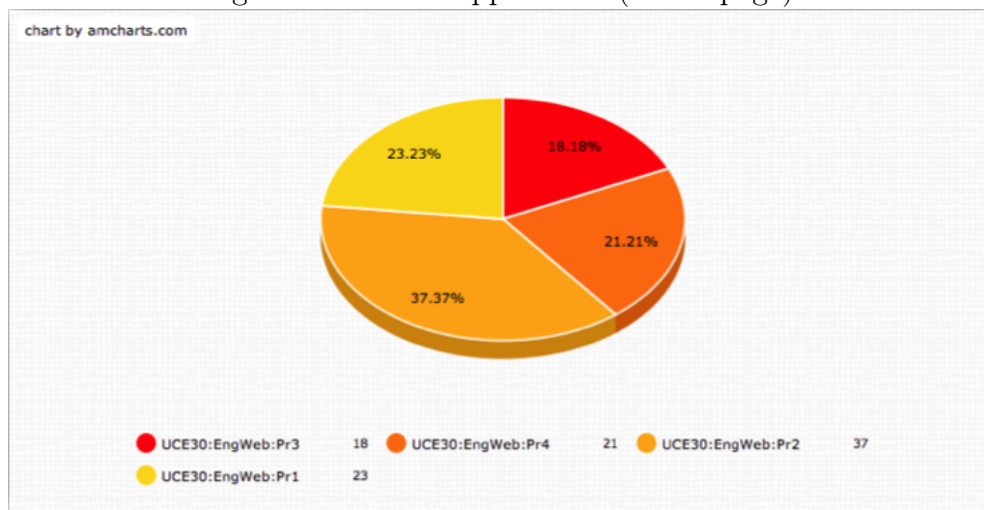


Figure 5.17: Test Application (Chart page).



## 5.5 Improvements

To produce logs with the most interesting information for my case study, some functionalities of the website had to be added or changed, as well as some minor changes in the original database. In the database, two new fields (maxkb and ndown) were added to the “Users” table. A new field (Spacekb) was also added to the table “Projectos”. The added functionalities to the application are:

- When trying to change the username of a user, the field “username” in the table “Project Records” also changes.
- When a user downloads a project, the field “nDown” on the “Users” table, reduces in 1 value. If the value is 0, the user cannot perform the download.
- Everyday, at 4 a.m, update the “nDown” field with a value of 5.
- Verify if the user has already exceeded his maximum limit of KB, if it happens it blocks any submission of reports by the user.
- A Producer can also delete projects.
- When the user changes his username, the name used on the logs also changes.

The work performed in connection with a test application, for the generation of fittest log’s was the development/creation of a repository in order to store and present projects/works of the students attending the minho university. The work was composed by four sections, projects (SIP intake and dissemination of a dip), users, statistics and logs, each one different features.

# Chapter 6

## PHP Infrastructure

FITTEST is non-invasive, and can log both high level and low level events. High level events are events that can be seen as produced by the users, whereas low-level events are events that tell us what happens inside a function execution as part of the target program's reaction to a high level event. These two classes of events require different logging approaches. To produce these logs, a PHP infrastructure would be needed. In this work, a library in PHP called `Fit_lib` was developed.

`Fit_lib` contains the class `serializer` with two important methods: the `serializeobj` and the `serializefield`, which serializes objects and the others types (String, Int, array, etc) respectively. The others important `fit_lib` functions are `logHighEvent` that belongs to high level events and `logFE`, `blockEvent` and `logFX` who belong to low level events. These functions are responsible to write the event into the `.log` file.

This logging solution is adaptable. It allows us to separately specify which events are to be logged, how to serialize them, and how we want to construct the abstraction of the target program's actual state.

Each target application has its own state, and for my test application (repository), I developed a `.php` file called `loger.php`. This file contains two classes (one for high level events and another for low level events) which can be fixed for others applications and one class that is specific for each application: the state class.

These three classes are responsible for serializing the fields/parameters. Because the 'event object', belonging to high level events and the low level events parameters are fixed, the classes can be used for any application, but it's necessary to create

the second part of the high level events, the state. Above it can be seen the state class for the test application.

```
class State
{
    public $php_page;
    public $user;
    public $name;
    public $type;
    public $kbspace;
    public $ndown;

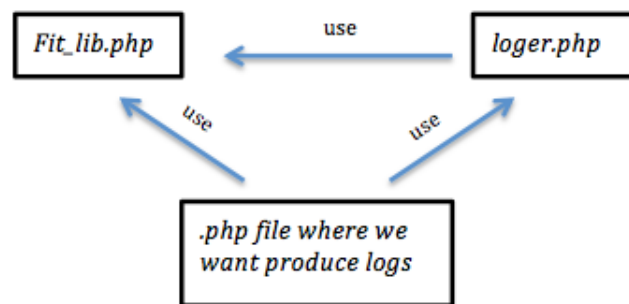
    function serializeself($serializer){
        $serializer->beginobject($this);
        $serializer->serializefield("php_page",$this->
php_page,0);
        $serializer->serializefield("user",$this->user,0);
        $serializer->serializefield("name",$this->name,0);
        $serializer->serializefield("type",$this->type,0);
        $serializer->serializefield("kbspace",
$this->kbspace,0);
        $serializer->serializefield("ndown",$this->ndown,1);
        $serializer->endobject($this);
    }

    public function __construct( $php_page, $user, $name, $type,
$kbspace, $ndown)
    {
        $this->php_page = $php_page;
        $this->user = $user;
        $this->name = $name;
        $this->type = $type;
        $this->kbspace = $kbspace;
        $this->ndown = $ndown;
    }
}
```

To use the library it's only necessary to include the `fit_lib.php` and `logger.php` files, create the objects for High or Low level events and call the respective function. Above it can be seen one example of how to produce a high level event.

```
$Flog = new HighLevelEvent("home", "homepage");  
$state = new State(page_name(), get_user(), get_user_name()  
, NULL, get_kbspace(), get_ndown());
```

To help to understand how the library works, a diagram was created and is shown below.



This infrastructure can be applied on any web application. In this particular case, it was been applied in the test application (repository) in order to produce FITTEST logs that can be analyzed. To do so, the application test was available online for a month and some users used the application to generate logs as more realistic as possible. As many information the logs file has, more accurate will be the analysis results. In the next chapter will be discussed the logs analysis, the used methods to try answer some key questions.





# Chapter 7

## Analising the FITTEST logs

After applying the PHP infrastructure on the test application (repository) it was hosted online so that it would be possible for different people to navigate on it and produce logs in a more realistic way.

After the log production, the focus was on analysis of the logs, and some key research questions have been formulated and should be answered:

- Q1 -> Can I identify the type (Administrator, Producer, Consumer) of an user?;
- Q2 -> Can I categorize the pages and the events based, e.g. on the types of users?
- Q3 -> Can I identify which pages are vulnerable to some types of attacks? For every type of event  $e$ , (e.g. SIP:Injection);
- Q4 -> Can I categorize the abstract states before (pre-state) and after (post-state) the events?

To answer the first question the conversion of a .log file to a .xml file has been made, and then Xpath was applied to find all the users on the logs, and for each user to read all the events related to him. The significant thing to do, to answer this question, is to verify the targetID related to a specific event performed by a unique user.

Imagine we have 6 different users: user1, user2, userA, userB, user $\alpha$  and user $\beta$ . User1 and user2 have 40 different targetIDs, userA and userB have 20 different targetIDs and user $\alpha$  and user $\beta$  have 10 different targetIDs. This can mean that we have 3 different groups of users with different permissions and can be useful to know how many types of users the application have.

The Xquery used to answer this question was:

- `distinct-values(//fd[@n="user"]//V//@v)` (Used to know the different users that are in the logs)
- `//fd[@n="user"]//V[@v="'tiago']/ancestor::E` (Used to know the events produced by the users, in this case the user "tiago")

To obtain an accurate value, It has been included a variable(factor), that was termed "factor S" (from space) which means the space, in number, of the total of TargetID of each one user. The factor S is a small percentage of the TargetID average and varies from 5 to 5 until 25 (in percentage). The maximum value of the "factor S" was limited to 25

The second question asks if it is possible categorize the pages and events based on user types founds on first question. To do that it's important to know which pages every user type accessed and split it, but before that the .xml was splitted by user and for each user the pages/events were grouped. After, each group of pages/events was checked and splitted for each user type.

Imagining that we have PageA.php, PageB.php, PageC.php, PageD.php and we have 2 users types founds on first question (User1 and User2). User1 accesses the pageA.php and PageB.php, the PageC only is accessed by User2 and PageD are accessed by all user types. In this case we have 3 categories of pages: pages accessed only by User1, pages accessed only by User2 and the last categorie pages accessed by all users. The same ideology for the events.

The Xquery used to answer this question was (for each user.xml):

- `//fd[@n='php_page']//V//@v` (Used to know the pages accessed)
- `//fd[@n='type']//V//@v` (Used to know the events accessed)

The first thing to do to answer the third question is make a research of pages vulnerabilities and type of attacks.

*"No language can prevent insecure code, although there are language features which could aid or hinder a security-conscious developer." - Chris Shiflett*

New flaws in web application security measures are constantly being researched, both by hackers and security professionals. Most of these flaws affect all dynamic web applications whilst others are dependent on specific application technologies. In both cases, one may observe how the evolution and refinement of web technologies also brings about new exploits which compromise sensitive databases, provide

access to theoretically secure networks, and pose a threat to the daily operation of online businesses. [?] In this application the focus centered on code injection (Sql injection, xml injection, xpath injection, cross-site scripting: XSS), default login and file inclusion vulnerability.

## Code injection

Code injection is a way to place software code into a computer system or program by exploiting unchecked assumed inputs. When a program assumes that only a certain input will occur and does not protect against different inputs being made, problems can occur. Code injection provides a way for hackers to gain access to data, modify data and corrupt code that they normally could not affect. [?]

To find potentials pages that can be vulnerable for code injection it's important to know, for example, which page have an input field. If a page has a input field this means that page can be attacked by hackers using code injection

The Xquery used to answer this question was:

- `//fd[@n="elem"]/ancestor::E//fd[@n="php_page"]//V//@v |  
//fd[@n="elem"]/ancestor::E//O[@ty="State"]//fd[@n="type"]//V//@v`  
(Used to know if a page have an input field)

## Default login

A default login is a kind of login, which is the same for every instance of the application. It's typically used to grant a first time access to hardware bundled control panels and administration interfaces.

To find pages vulnerable to this type of attack should be found which pages have a login system. If a page has a login system, maybe a login is a typical user: admin pass:admin or user:administrator pass:"name of application", so in this case that page are vulnerable to attack.

## File include vulnerability

The file inclusion attack is very similar to directory traversal attack. The only difference is that with directory traversal attack, we can only read the file we're not allowed to read, but with file inclusion we're including the file into the current web page execution, thus executing the file we're not allowed to execute. To find this kind of vulnerability on pages it's necessary to know which page has an upload system. The File upload facilities are also usually considered dangerous because

they can be abused to leverage various types of attacks.

In last two mentioned vulnerabilities (default login and file include) the Xpath and method to find the pages used was the same. The only difference was on regular expression used: words like “login” for default login and “upload” for file include.

- `//fd[@n='php_page']//V//@v` (Used to know the pages accessed)
- `//fd[@n='type']//V//@v` (Used to know the events accessed)

What can happen if a web application is vulnerable?:

- An attacker can output the contents of any php file raw to the browser, where he can possibly obtain sql login/password to your database.
- An attacker can use your website to send out large amounts of spam to various email addresses.
- An attacker can deface your website.
- An attacker can obtain private information.
- An attacker may gain access to the whole server.

The last question centers on possibility to categorize the state before (pre-state) and after (post-state). To make categorization should take into account the occurrence of each event, therefore four categories were created:

- Very unlikely (0-25%)
- Unlikely (26%-50%)
- Likely (51%-75%)
- Very likely (76%-100%)

To do the categorization, first it's necessary to extract all event types of users, then for each type not only the next log event (or before log event) should be checked but also the occurrence of each event be considered to decide which category the occurrence belongs.

The Xquery used to answer this question was:

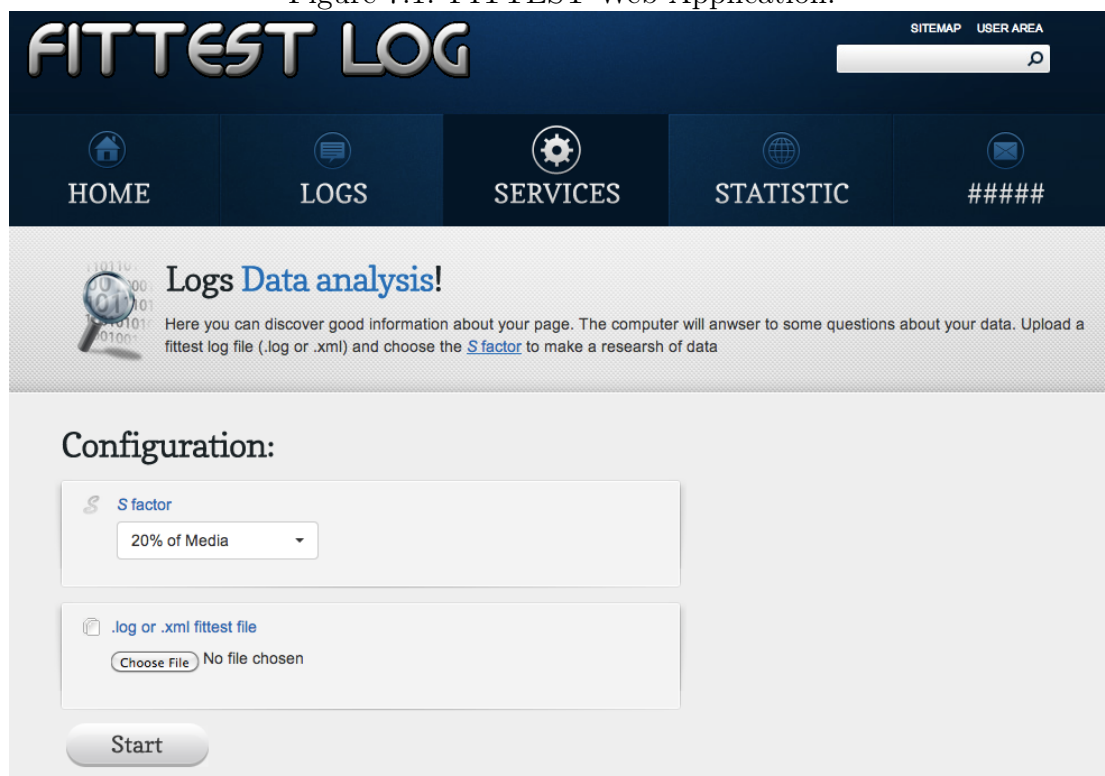
- `//0[@ty="State"]/fd[@n="type"]//V//@v` (Used to know all the event types)

- `//0[@ty="State"]/fd[@n="type"]/V[@v=\'SIP:injection\']/following::E[1]`  
`/0[@ty="State"]/fd[@n="type"]/V/@v` (Used to know the next log event of, in this case, 'SIP:injection' event type).

To interactively answer these questions, a web site was created. The users only need to upload the .log file or .xml file (converted by *haslog*), choose the percentage of factor S and the application will try answer the questions.

The web application can be seen in figure 7.1.

Figure 7.1: FITTEST Web Application.



The screenshot shows the FITTEST LOG web application interface. At the top, there is a dark blue header with the title "FITTEST LOG" in large, stylized letters. To the right of the title, there are links for "SITEMAP" and "USER AREA", and a search bar. Below the header is a navigation menu with five items: "HOME" (with a home icon), "LOGS" (with a speech bubble icon), "SERVICES" (with a gear icon), "STATISTIC" (with a globe icon), and "#####" (with an envelope icon). The main content area has a light gray background. It starts with a section titled "Logs Data analysis!" featuring a magnifying glass icon over binary code. Below this title is a paragraph: "Here you can discover good information about your page. The computer will answer to some questions about your data. Upload a fittest log file (.log or .xml) and choose the [S factor](#) to make a research of data". Below this is a "Configuration:" section with two input fields. The first field is labeled "S factor" and has a dropdown menu currently showing "20% of Media". The second field is labeled ".log or .xml fittest file" and contains a "Choose File" button and the text "No file chosen". At the bottom of the configuration section is a "Start" button.

When we upload a .log file, the application will analyse the data and answer a question's overview and some options for each question will be provided with some extra information (figure 7.2).

Figure 7.2: Answers overview

### It is possible identify the type of an user?

In a total of 10 users founds in logs, with a *S factor* = 0.2, the induced number of type users was 3 type(s):

1° type: 8 users

---


2° type: 1 users

---


3° type: 1 users

---

#### Options

-  User List

---

-  view graph

### It is possible categorize the pages and the events based on types of users?

Induced number of categories for pages based on 3 types(s) users was 5 categories(s):

1° categorie: 1 pages accessed by only 2° user's type

---

2° categorie: 4 pages accessed by only 3° user's type

---

3° categorie: 2 pages accessed by 1° and 3° user's type

---


4° categorie: 5 pages accessed by 2° and 3° user's type

---


5° categorie: 5 pages accessed by all user's type

---


#### Options

-  Pages Categories


---

-  Pages Diagram

---

-  Events Categories

---

-  Events Diagram

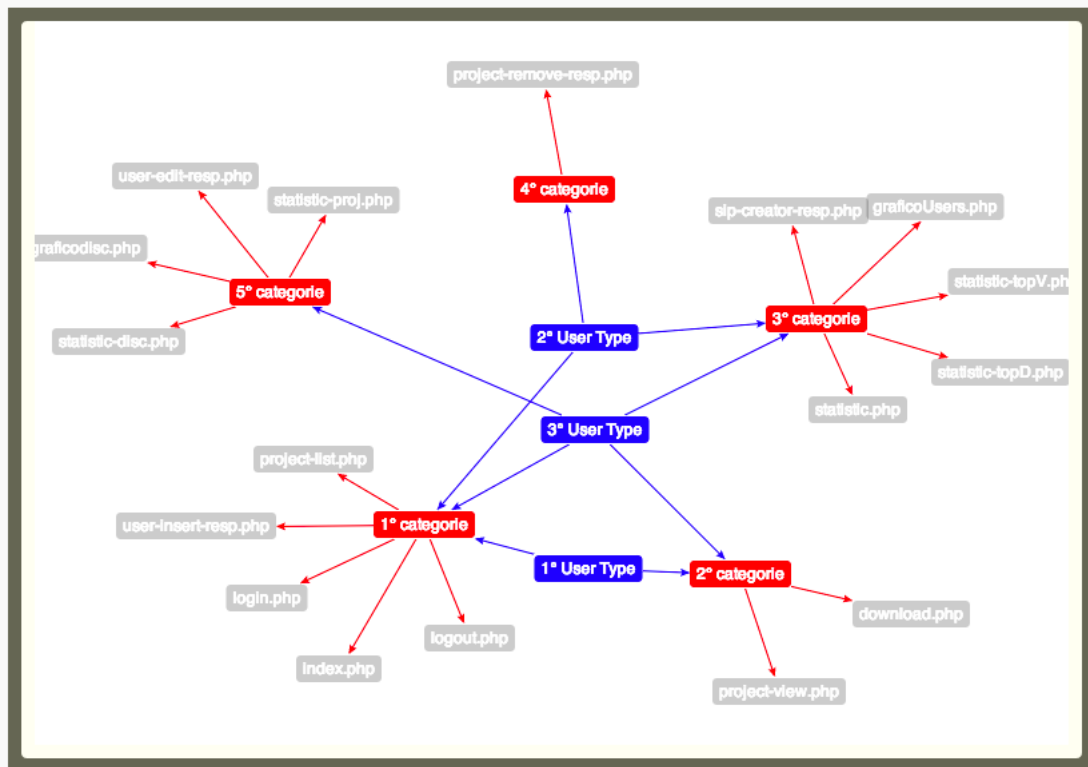
In first question we can list the names of users who belong to each user type and generate a graphic to see the different groups of TargetID's which means a different user types. (figure 7.3)

Figure 7.3: User types Groups

<b>1° type: 8 users</b>
<b>2° type: 1 users</b>
<i>admin</i>
<b>3° type: 1 users</b>

In second question we can list the pages or events by categories and see what user/users can access to that page or event and a page/event diagram to have an overview of the categories (figure 7.4).

Figure 7.4: Categories overview



In third question we only have one option that enables listing all pages and respective events of the application as being vulnerable for different type of attacks. We can view an example on figure 7.5.

Figure 7.5: List of vulnerabable pages

<b>Code Injection :: 8 Pages</b>
<i>sip-creator-resp.php -&gt; SIP:Ingestion-form</i>
<i>statistic.php -&gt; Statistic:View</i>
<i>statistic-topD.php -&gt; Statistic:View</i>
<i>statistic-topV.php -&gt; Statistic:View</i>
<i>user-insert-resp.php -&gt; User:Add</i>
<i>user-edit-resp.php -&gt; User:Edit</i>
<i>statistic-proj.php -&gt; Statistic:View</i>
<i>statistic-disc.php -&gt; Statistic:View</i>
<b>Default Login :: 1 Pages</b>
<b>File Injection :: 1 Pages</b>



For final question we can list the Pre/Post-states of every event type and category by the probability of that happens (figure 7.6 and 7.7).

Figure 7.6: Post-state by groups

<b>Post-state :: very likely (3)</b>
<i>User:Logout -&gt; User:Login :: 93.75%</i>
<i>DIP:Remove -&gt; DIP:List :: 100%</i>
<i>Graphic:View -&gt; Statistic:View :: 95.24%</i>
<b>Post-state :: likely (5)</b>
<b>Post-state :: unlikely (8)</b>
<b>Post-state :: very unlikely (45)</b>

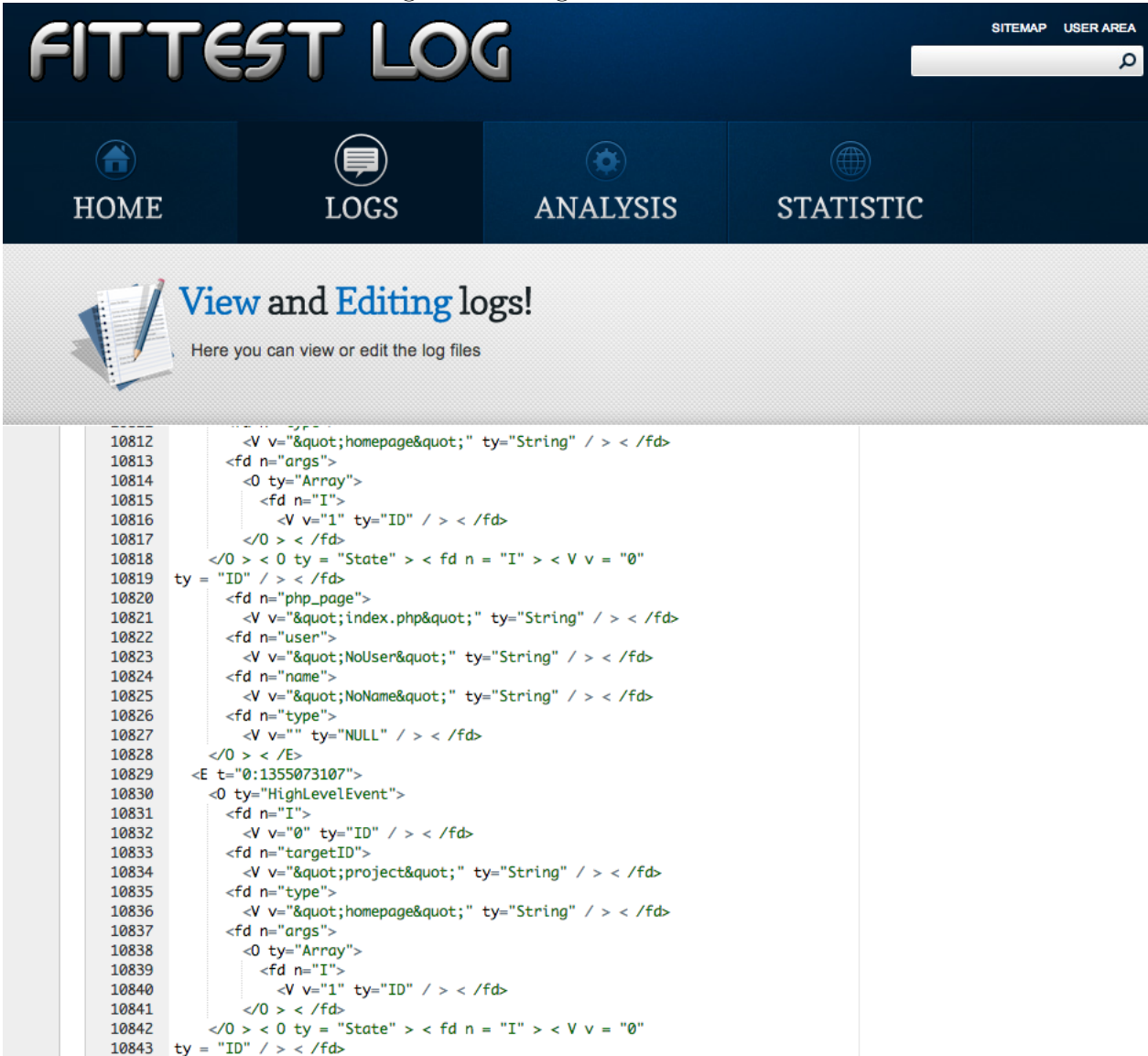
Figure 7.7: Post-state of all events

<b>Post-state: User:Add</b>
<i>User:Login -&gt; likely :: 54.55%</i>
<i>User:Logout -&gt; very unlikely :: 18.18%</i>
<i>Statistic:View -&gt; very unlikely :: 9.09%</i>
<i>DIP:List -&gt; very unlikely :: 18.18%</i>
<b>Post-state: User:Login</b>
<b>Post-state: DIP:List</b>
<b>Post-state: User:Logout</b>
<b>Post-state: DIP:Search</b>
<b>Post-state:</b>
<b>Post-state: DIP:View</b>
<b>Post-state: DIP:Download</b>
<b>Post-state: SIP:Ingestion-form</b>
<b>Post-state: DIP:Remove</b>
<b>Post-state: Statistic:View</b>

## Logs Editor

The analysis application have an editor functionality, so users can view or edit FITTEST .log or .xml files.( Figure 7.8). The input form look like the same as services (Figure 7.1)

Figure 7.8: Logs Editor



The screenshot displays the FITTEST LOG application interface. At the top, the title "FITTEST LOG" is prominently displayed in a stylized font. To the right, there are links for "SITEMAP" and "USER AREA", and a search bar. Below the title, a navigation menu contains four main sections: "HOME", "LOGS", "ANALYSIS", and "STATISTIC". The "LOGS" section is currently selected and highlighted. Below the navigation menu, a banner area features a notebook icon and the text "View and Editing logs!" followed by the subtitle "Here you can view or edit the log files". The main content area shows a list of log entries with their corresponding XML data. The visible XML code includes elements like <V v="&quot;homepage&quot;" ty="String" />, <fd n="args">, <0 ty="Array">, <fd n="I">, <V v="1" ty="ID" />, </0 > </fd>, </0 > <0 ty = "State" > < fd n = "I" > < V v = "0" ty = "ID" /> </fd>, <fd n="php\_page">, <V v="&quot;index.php&quot;" ty="String" />, <fd n="user">, <V v="&quot;NoUser&quot;" ty="String" />, <fd n="name">, <V v="&quot;NoName&quot;" ty="String" />, <fd n="type">, <V v="" ty="NULL" />, </0 > </E>, <E t="0:1355073107">, <0 ty="HighLevelEvent">, <fd n="I">, <V v="0" ty="ID" />, <fd n="targetID">, <V v="&quot;project&quot;" ty="String" />, <fd n="type">, <V v="&quot;homepage&quot;" ty="String" />, <fd n="args">, <0 ty="Array">, <fd n="I">, <V v="1" ty="ID" />, </0 > </fd>, </0 > <0 ty = "State" > < fd n = "I" > < V v = "0" ty = "ID" /> </fd> ..

All information shown by analysis application can be useful for many things. For example, the first two questions reveal that there are three user types and a list of users which were categorized as one of these user types. If a user with administrator privileges has been categorized as a regular user, it means that this user may not need of the administrator privileges. The third question alerts developers which pages can be targeted of any type of injection and with last question it is possible

to know the sequence of pages that usually users follow.

# Chapter 8

## Conclusion

*The greater the obstacle, the more glory in overcoming it.*

*Molière*

This work allowed the development of an infrastructure for PHP for generating logs in format FITTEST. The developed infrastructure has been implemented on web application created by us. This application consisted in projects and academic works storage in order to organize them in a virtual repository enabling to check and see it any time. To verify the correct implementation of the infrastructure developed was asked to ten students to they used the web application in order to produce the logs with more possible veracity.

This work aimed the study of the potential of the FITTEST logs and four key questions were generated: Can the type of user be identified? Can the pages and the events categorized, e.g. on the types of users? Can be identified which pages are vulnerable to some types of attacks? Can the abstract states before (pre-state) and after (poststate) be categorized? The results after the analysis utilizing Xpath were satisfactory since the four key questions have been answered successfully. The biggest difficulty in performing this work was in development of auto-serialization of logs in the infrastructure.

In conclusion, the FITTEST logs can be a great contribute for internet future and mainly for web's applications. After analysis, it is possible predict the variety of users and then categorize the pages and events based on users, to identify pages that can be target of code injection attacks or even categorize the before/after events state.

For future work are suggested improvements on infrastructure in order to obtain more information and perform news logs analysis by formulation of new questions and try to answer them. A very interesting improvement could be the application of data minning, specifically the clustering technique to answer the question: "Can I identify the user type (ex. Administrator, producer and consumer) of a web application?"

# Bibliography

- [1] Prasetya, W., Middelkoop, A., Elyasov A., and Hage, J. (2011) D6.1: FITTEST Logging Approach. Project no. 257574, FITTEST Future Internet Testing.
- [2] Prasetya, W., Middelkoop, A., Elyasov A., and Hage, J. (2011) FITTEST Log Format (version 1.1)
- [3] Oficina da Net [online] [seen on September 2013]. Accessed on World Wide Web: < [http://www.oficinadanet.com.br/artigo/2227/mysql\\_-\\_o\\_que\\_e](http://www.oficinadanet.com.br/artigo/2227/mysql_-_o_que_e)>
- [4] Criar Web [online] [seen on September 2013]. Accessed on World Wide Web: < <http://www.criarweb.com/artigos/202.php>>
- [5] Oficina da Net [online] [seen on September 2013]. Accessed on World Wide Web:< [http://www.oficinadanet.com.br/artigo/659/o\\_que\\_e\\_php](http://www.oficinadanet.com.br/artigo/659/o_que_e_php)>
- [6] Criar Web [online] [seen on September 2013]. Accessed on World Wide Web: < <http://www.criarweb.com/artigos/184.php>>
- [7] Tec Mundo [online] [seen on September 2013]. Accessed on World Wide Web: <<http://www.tecmundo.com.br/1762-o-que-e-xml-.htm>>
- [8] Oxygen xml [online] [seen on September 2013]. Accessed on World Wide Web: <<http://www.oxygenxml.com/>>
- [9] EPCglobal (2005) The Application Level Events (ALE) Specification (version 1.0)
- [10] Luckham,D., Frasca, B. (1998) Complex Event Processing in Distributed Systems
- [11] GWT create [online] [seen on July 2013]. Accessed on World Wide Web: <<http://www.gwtproject.org/doc/latest/DevGuideLogging.html>>

- 
- [12] Typo3 - Aspect-Oriented Programming [online] [seen on July 2013]. Accessed on World Wide Web: <<http://docs.typo3.org/flow/TYPO3FlowDocumentation/TheDefinitiveGuide/PartIII/AspectOrientedProgramming.html>>
- [13] Thought Forge - Creating a Logging Aspect with AOP [online] [seen on July 2013]. Accessed on World Wide Web: <<http://thoughtforge.net/665/creating-a-logging-aspect-with-spring-aop-and-aspectj/>>
- [14] Log File Formats [online] [seen on July 2013]. Accessed on World Wide Web: <<http://publid.boulder.ibm.com>>
- [15] Nihuo Software (log analyzer) [online] [seen on July 2013]. Accessed on World Wide Web: <<http://www.loganalyzer.net/log-analyzer/w3c-extended.html>>
- [16] Extended Log File Formats [online] [seen on July 2013]. Accessed on World Wide Web: <<http://www.w3.org/TR/WD-logfile-960221.html>>
- [17] Gulcu, C. (2000) "Log4j delivers control over logging," Java World.
- [18] Apache log4j - tutorialspoints [online] [seen on August 2013]. Accessed on World Wide Web: <[http://www.tutorialspoint.com/log4j/log4\\_patternlayout.htm](http://www.tutorialspoint.com/log4j/log4_patternlayout.htm)>
- [19] Labeled Tab-separated Values [online] [seen on August 2013]. Accessed on World Wide Web: <<http://ltsv.org/>>
- [20] Syslog [online] [seen on August 2013]. Accessed on World Wide Web: <<http://www.syslog.org/>>
- [21] Schuster, A. (2007) Introducing the microsoft vista event log file format. Digital investigation, 4:65-72.
- [22] Boroday, S., Hallal, H., Petrenko, A. and Ulrich, A. (2003) Formal Modeling of Communication Traces. In *ISTA*, pages 97-108.
- [23] Jain, S., Singh, I., Chandra, A., Zhang, Z. and Bronevetsky, G. (2009) Extracting the textual and temporal structure of supercomputing logs, pages 254-263.
- [24] Liang, Y., Zhang, Y., Jette, M., Sivasubramaniam, A. and Sahoo, R. (2006) Bluegene/L failure analysis and prediction models, pages 425-434.
- [25] Andrew, J. (1998) Testing using log file analysis: tools, methods and issues. In *13th IEEE International Conference on Automated Software Engineering*.

- 
- [26] Andrew, J., Na, B. (1998) Theory and practice of log file analysis. Technical report, Dept. of Computer Science, Univ. of Western Ontario.
- [27] Andrew, J. and Zhang, Y. (2000) Broad-spectrum studies of log file analysis, pages 105-114.
- [28] Yantzi, D. and Andrew, J. (2007) Industrial evaluation of a log file analysis methodology. In *Proceedings of the 5th International Workshop on Dynamic Analysis*.
- [29] Tu, D., Chen, R., Liu, Y. and Du, Z. (2009) A Method of Log File Analysis for Test Oracle.
- [30] Cotroneo, D., Pietrantuono, R., Mariani, L. and Pastore, F. (2007) Investigation of failure causes in workload-driven reliability testing, pages 78-85.
- [31] Feather, M. (1998) Rapid application of lightweight formal methods for consistency analyses
- [32] Feather, M. and Smith, B. (1999) Automatic generation of test oracles - from pilot studies to application, pages 12-15.
- [33] Baresi, L. and Young, M. (2001) Test oracles. Technical report
- [34] Robinson, W. (2002) Monitoring software requirements using instrumented code
- [35] Ducasse, D., Girba, T. and Wuyts, R. (2006) Object-oriented legacy system trace-based logic testing
- [36] Kowalski, K. and Beheshti, M. (2006) Improving security through analysis of log files intersections. Accessed on World Wide Web: <<http://ijns.femto.com.tw/contents/ijns-v7-n1/ijns-2008-v7-n1-p24-30.pdf>>
- [37] Ernst, M., Cockrell, J., Griswold, W. and Notkin, D. (2001) Dynamically discovering likely program invariants to support program evolution.
- [38] Ernst, M., and Nimmer, J. (2002) Automatic generation of program specifications. pages 232-242
- [39] Barringer, H., Groce A., Havelund, K. and Smith, M. (2010) Formal analysis of log files.



- 
- [40] Lorenzoli, D., Mariani, L. and Pezzè, M (2008) Automatic generation of software behavioral models. Accessed on World Wide Web: <<http://www.lta.disco.unimib.it/lta/uploads/papers/LorenzoliEtAl-GKTail-ICSE-2008.pdf>>
- [41] Ruuskanen, Aleksi (2013) IBM Coremetrics - Web analytics and Digital Marketing Optimization
- [42] Taksa, I., spink, A., and Jansen, B. Web log analysis: Diversity of Research Methodologies
- [43] Analog2.11 [online] [seen on August 2013]. Accessed on World Wide Web: <[http://scholar.lib.vt.edu/www\\_stats/borg/analog.html](http://scholar.lib.vt.edu/www_stats/borg/analog.html)>
- [44] webtrends [online] [seen on August 2013]. Accessed on World Wide Web: <<http://webtrends.com/>>
- [45] Kuashik, Avinash (2007). Web Analytics: An hour a day.
- [46] Grace, L.k., Maheswari, V., Nagamalai, D. (2011) Analysis of web logs and web user in web minning
- [47] WebTrends Inc. (July 2009 Edition) WebTrends Analytics Software Implementation and Maintenance Guide. Accessed on World Wide Web: <[https://www.heureka.com/upload/WebTrendsAnalyticsSoftware\\_ImplementationGuide.pdf](https://www.heureka.com/upload/WebTrendsAnalyticsSoftware_ImplementationGuide.pdf)>
- [48] WebTrends Inc. (April 2006 Edition) WebTrends Analytics Software Quick Start Guide. Accessed on World Wide Web: <[http://www.waomarketing.com/documents/WebTrends\\_Software\\_Quick\\_Start\\_Guide.pdf](http://www.waomarketing.com/documents/WebTrends_Software_Quick_Start_Guide.pdf)>
- [49] Omniture SiteCatalyst Review [online] [seen on September 2013]. Accessed on World Wide Web: <<http://media.about.com/od/newmediatools/gr/Omniture-Sitecatalyst.htm>>
- [50] Adobe Insight, powered by Omniture [online] [seen on September 2013]. Accessed on World Wide Web: <[http://www.adobeeventscanada.com/aga2012/collateral/09\\_whitepaper\\_insight\\_government2.pdf](http://www.adobeeventscanada.com/aga2012/collateral/09_whitepaper_insight_government2.pdf)>

- 
- [51] IBM - Web analytics [online] [seen on September 2013]. Accessed on World Wide Web: <<http://www-01.ibm.com/software/marketing-solutions/web-analytics/>>
- [52] Urchin [online] [seen on September 2013]. Accessed on World Wide Web: <<http://www.google.com/urchin/index.html>>
- [53] Urchin Tools [online] [seen on September 2013]. Accessed on World Wide Web: <<http://www.urchintools.com/>>
- [54] Piwik [online] [seen on September 2013]. Accessed on World Wide Web: <<http://piwik.org/>>