AUTOMATING THE TESTING PROCESS OF IMAGE PROCESSING ALGORITHMS

SUBMITTED TO THE SCHOOL OF BUSINESS AND HUMANITIES

OF INSTITUTE OF TECHNOLOGY, SLIGO

FOR THE AWARD OF

MASTER OF SCIENCE IN COMPUTING BY RESEARCH

JOHN BRADY

2006

ABSTRACT

As digital imaging processing techniques become increasingly used in a broad range of consumer applications, the critical need to evaluate algorithm performance has become recognised by developers as an area of vital importance. With digital image processing algorithms now playing a greater role in security and protection applications, it is of crucial importance that we are able to empirically study their performance. Apart from the field of biometrics little emphasis has been put on algorithm performance evaluation until now and where evaluation has taken place, it has been carried out in a somewhat cumbersome and unsystematic fashion, without any standardised approach. This paper presents a comprehensive testing methodology and framework aimed towards automating the evaluation of image processing algorithms. Ultimately, the test framework aims to shorten the algorithm development life cycle by helping to identify algorithm performance problems quickly and more efficiently.

ACKNOWLEDGMENTS

I would like to take this opportunity to thank a number of people who have contributed to the completion of this thesis. Firstly, I want to give special thanks to my research supervisor Dana Vasiloaica for her clear advice, help and encouragement during the course of the project. I would also like to thank my research colleague Colin Callanan for his help, cooperation and support over the last two years. Thanks also to my work colleagues at the partner company who I worked with at various times during the course of the project. Much of this work was carried out in the Research Office of the Department of Business and Humanities of the Institute of Technology, Sligo and I wish to thank all my colleagues there for creating such an enjoyable and stimulating environment for conducting research work. Last but not least I am grateful to my parents Noel and Christina, my brother Paul and all my family and friends for their support, guidance and encouragement over the course of the last two years.

DECLARATION

TO WHOM IT MAY CONCERN

The work in this thesis "Automating the testing process of image processing algorithms" represents the research carried out by John Brady under the supervision of Dana Vasiloaica, and does not include work by any other party, with acknowledged exception.

Signed: John Brady

Date: 24/09/06

TABLE OF CONTENTS

LIST OF TABLES

Number *Page*

LIST OF FIGURES

Number *Page*

GLOSSARY OF ABBREVIATIONS

API	Application Programming Interface
AWT	Abstract Windowing Toolkit
BSF	Bean Scripting Framework
DLL	Dynamic Link Library
FERET	Face Recognition Technology
FRVT	Face Recognition Vendor Test
GUI	Graphical User Interface
HCI	Human Computer Interaction
IDC	International Data Corporation
IDE	Integrated Development Environment
IP	Image Processing Algorithm
IT	Information Technology
JAXP	Java API for XML Processing
JNI	Java Native Interface
JPEG	Joint Photographic Experts Group
JRE	Java Runtime Environment
MFC	Microsoft Foundation Classes
MIT	Massachusetts Institute of Technology
MVC	Model View Controller
NASA	National Aeronautics and Space Administration
OS	Operating System
PEIPA	Pilot European Image Processing Archive
PFML	Photographic Feature Markup Language
PIE	Pose, Illumination and Expression variant
RCP	Rich Client Platform

SDLC	Software Development Life Cycle
SWT	Standard Widget Toolkit
UI	User Interface
UML	Unified Modelling Language
U.S.	United States
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XPath	XML Path language

CHAPTER 1: INTRODUCTION

Introduction

1.1 INTRODUCTION

The last 10 years have seen a digital image revolution, with soaring interest in image processing technology across the consumer and business landscape. The proliferation of digital cameras and mobile camera phones in today's world has resulted in a phenomenal surge in the use of consumer digital images. (Eastman Kodak, 2006) a digital imaging market leader reports that we're moving picture viewing and sharing from an occasional experience to an always on lifestyle experience giving consumers the ability to be connected to people and pictures regardless of place or time. Central to the adoption of digital photography by both businesses and consumers have been advances in the areas of image processing and image analysis, particularly in the development of complex image processing algorithms.

Digital image processing is a collection of techniques/algorithms for the manipulation of digital images by computers. Standard features on most digital cameras sold nowadays, such as automatic red eye removal, are the result of complex mathematical operations known as algorithms. Without realising it, users of everyday digital consumer appliances such as digital cameras, software graphics packages and printers are harnessing years of research and development in image processing technology.

Of course use of image processing algorithms have not been confined to individual consumer appliances like digital cameras, but have driven advancements in many scientific practices, stretching from medicine to security. From a security viewpoint the

need to duplicate human vision has been one of the main driving forces behind developments in image processing.

As digital image processing techniques become increasingly used in a broad range of applications (Gonzalez & Woods, 2002:6, Jaynes et al, 2005:1), the critical need to evaluate algorithm performance has become recognised by developers as an area of vital importance (Courtney & Thacker, 2001; Hua et al, 2004:498; Meer et al, 2000:2 Micheals & Boult, 2001:150). Proper evaluation has always been very important for any research area but the field of image processing currently lacks a comprehensive testing framework for assessing the performance of image processing algorithms. Only with a correct and standardised evaluation can advances in the field be identified and encouraged. With algorithms being developed without their likely performance being calculated beforehand, this often leads to a somewhat ad hoc approach to development. This thesis proposes to investigate and develop software tools to facilitate the evaluation of these image processing algorithms on large datasets of consumer digital images.

1.3 PROJECT OVERVIEW

THESIS TITLE: Automating the testing process of image processing algorithms.

This thesis, the result of a two years masters degree programme, presents a software solution aimed at automating the testing process of image processing algorithms. The aims of the research are to:

1. Study and gain expertise in the area of image processing techniques in general. Having an understanding of how image processing algorithms operate is of great importance before attempting to test the algorithms.

2. Research into existing methodologies used to test software in general, and image processing algorithms in particular. Conclude by identifying the best approach concerning testing image processing algorithms.
3. Research and develop a methodology for automatic testing of image processing algorithms. This takes into consideration particularities of algorithm testing (i.e. version control) and reporting (i.e. allow for inspection of images that the algorithm did process correctly).
4. Become proficient in using the programming techniques required to implement the testing application. These techniques include:
    - User Interface (UI) Design paying particular attention to Graphical User Interfaces (GUI) suitable for testing applications and reporting.
    - Design Patterns.
    - Java Development using the Eclipse Integrated Development Environment (IDE).
    - EXtensible Markup Language (XML) technology.
5. Design and implement a database to store the results of testing various image processing algorithms.
6. Develop user friendly tools to support the testing of various image processing techniques.
7. Implement techniques which allow for image comparison. An algorithm is tested by executing it on a set of relevant, previously marked images. As a result, every image in the set is modified in some way, hopefully in the marked areas. Comparison between the original images and the modified images has to be done to conclude if the algorithm worked correctly.
8. Develop statistical tools to analyse the results of algorithm testing. The results from different versions of the same algorithm can then be compared against each other.

1.4 THESIS OUTLINE

1.4.1 CHAPTER 1: INTRODUCTION

This chapter has discussed some of the background to the research components of this thesis. The chapter commenced with an introduction to the increasing use of digital image processing algorithms in everyday consumer applications and the critical need to evaluate these algorithms. A brief overview of the history and purpose of the "*Automating the testing process of image processing algorithms*" project was then presented along with the projects key aims and objectives. Finally an outline of the thesis structure is presented.

1.4.2 CHAPTER 2: LITERATURE REVIEW

The second chapter gives an overview of the image processing industry stressing the importance of image processing algorithm testing and identifies the need for an efficient framework to address testing needs. The chapter starts by introducing image processing terminology and algorithms, and then moves on towards identifying problems associated with algorithm development and testing. Finally the core requirements of a comprehensive image processing algorithm testing framework are detailed to further assess and improve the quality of existing and newly developed algorithms.

1.6.3 CHAPTER 3: REQUIREMENTS ANALYSIS

The third chapter outlines the aims and objectives of the proposed testing methodology, a solution to current testing and evaluation difficulties. The requirements of a software solution are outlined in detail and the set of key functional requirements are presented. Finally a review of the key problems the proposed testing framework intends to solve is presented.

1.6.4 Chapter 4: Design

The fourth chapter draws up a blueprint for system implementation called the design. Firstly, the overall algorithm test framework is described and the classification of the two distinct categories of testing tool user is explained. The key non-functional requirements based on the two categorises of user for the testing tool are then identified. After the selection of system development life cycle and technologies are explained the next sections outline the main design aspects of the testing tool. Firstly design of the testing tool algorithm integration architecture is explained. The subsequent section then describes the design of the components that make up a test scenario and the underlying test execution architecture, while the penultimate section details the GUI design process. The concluding section summarises how the chosen design satisfies each of the initial requirements defined in the previous chapter.

1.6.5 Chapter 5: Implementation

The fifth chapter focuses on describing some of the important implementation techniques used during the development of the testing tool. The chapter commences by providing an overview of the overall algorithm test framework. With a similar approach to the design chapter, the next sections describe the individual stages of the testing tool's implementation. Firstly the development of the components that make up a test scenario and the underlying test execution architecture is explained, in particular the process of writing test scripts incorporating relevant metrics of interest to analyse algorithm performance. The subsequent section then provides an overview of the overall image testing application UI, while the final section details the GUI of the testing tool. In addition code snippets are provided throughout this chapter to give a deeper understanding of how the testing tool really works.

1.6.6 Chapter 6: Testing and Evaluation

The sixth chapter summarises the testing techniques used to evaluate the software solution including both the formative and summative evaluation techniques utilised.

1.6.7 Chapter 7: Conclusion and Further Work

In the seventh chapter the conclusion is presented along with suggestions for further improvements to the software solution into the future.

CHAPTER 2: LITERATURE REVIEW

Literature Review

2.1 INTRODUCTION

The chapter starts by introducing the everyday use of digital image processing technology in consumer appliances like digital cameras and its increasing importance to other areas of industry including security. After a selective review of popular image processing algorithms used in both consumer appliances and computer vision applications, the main problems in image processing algorithm development and testing are presented. A comprehensive survey on the methods and techniques being used in image processing algorithm performance assessment is then undertaken and the need for an efficient image processing algorithm testing framework is identified. Finally, the main components of a comprehensive image processing algorithm testing methodology that will aid in assessing and improving the quality of existing and newly developed algorithms are outlined.

2.2 DIGITAL IMAGE PROCESSING OVERVIEW

2.2.1 EVERYDAY USE OF DIGITAL IMAGE PROCESSING TECHNOLOGY

"We are in the midst of a revolution sparked by rapid progress in digital image processing technology." (Gur, 2002)

(Milburn, 2004; Smolka et al, 2003; Wilhelm et al, 2004; Zhang et al, 2002; Zhang et al, 2004a) Today the world is in a midst of a digital imaging revolution as digital technology replaces traditional photography and high-tech digital image processing techniques become available in consumer software and photographic equipment (Gur, 2002; Bovik, 2000). Technological advances and the convergences of digital imaging and wireless

technologies have brought many changes in the way digital images are captured, manipulated, analysed, transmitted and printed.

"*One aspect of image processing that makes it such an interesting topic of study is the amazing diversity of applications that use image processing or analysis techniques*" (Bovik, 2000:3)

The revolution has not only affected individual lifestyle habits such as the way families interact through camera phones but have influenced all areas of science from tumor detection in biomedicine, to monitoring of weather patterns in environmental science and object and scene perception in robotic vision. (Gur, 2002) reports image processing as one of the most rapidly evolving areas of Information Technology (IT) with growing applications in all areas of businesses.

According to (Narayanaswami & Raghunath, 2004:67) digital photography has changed our entire photo experience for the better. Digital cameras have eliminated the need for film as the image is digitally captured and stored in a memory array within the camera allowing photos to be viewed and enjoyed virtually instantaneously (Bovik, 2000). And since they first appeared on the scene in the mid 1980's digital cameras have changed from complicated and expensive tools of limited value to user-friendly, cheap, powerful and effective tools that can be used in a wide array of tasks (Davis et al, 2005; Gargi et al, 2003; Girgensohn et al, 2004a; Milburn, 2004; Van House et al, 2005). Supported by advanced image processing (IP) algorithms they have become an ubiquitous and requisite commodity in the modern technological age for the recording, displaying and communication of visual representations (Messina et al, 2003:549). For instance red eye continues to be the most common customer complaint in the digital imaging market (Luo et al, 2004; Schettini et al, 2004:139; Smolka et al, 2003:1767; Zhang (b) et al 2004) and most digital cameras sold today incorporate red-eye filters which analyse the captured image for the red-eye phenomenon and correct the image by changing the red area to its original colour.

"*Virtually everyone is in some way affected by personal photography – as photographer, subject, or viewer.*" (Van House et al, 2005:1853)

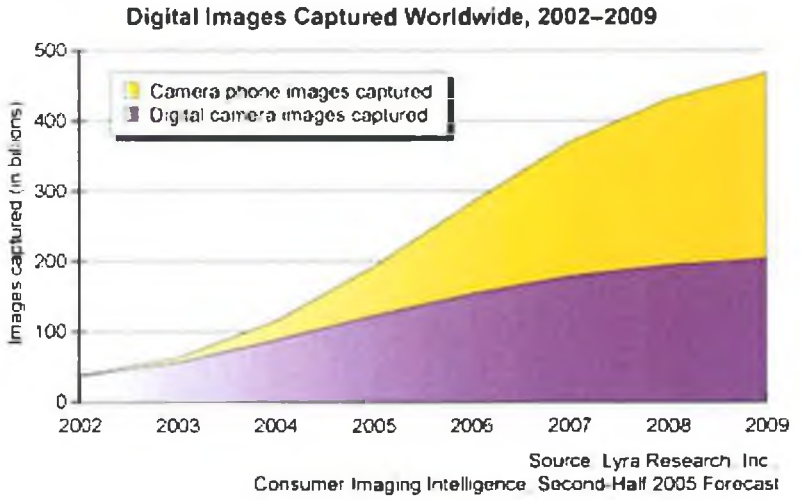


Figure 1: Digital Images Capture Worldwide Forecast

The whole concept of how we share digital photographs is changing as online sharing, digital archives, email and picture messaging transform the whole viewing experience (Wilhelm et al, 2004:1406). (Eastman Kodak, 2006) A digital imaging market leader reports that we're moving picture viewing and sharing from an occasional experience to an always-on lifestyle experience giving consumers the ability to be connected to people and pictures regardless of place or time. Consequently as (Girgensohn et al, 2003; Sarvas et al, 2004; Sarvas, 2005; Wilhelm et al, 2004:1403; Zhang et al, 2004a) report, there has been a phenomenal surge in use of consumer digital images with (International Data Corporation, 2004), a market leader in research and consulting, predicting that digital camera images captured, shared and received worldwide will grow an average of 35% from 2003 to 2008.

"*By reducing many of the barriers to camera phone use and image sharing (including increasing image quality, easing the sharing process, and removing barriers), we find that users quickly develop new uses for imaging.*" (Van House et al, 2005:1853)

Given the increasing use of digital image processing technology in a wide array of both consumer and industrial applications, testing the accuracy this technology becomes of

even greater importance to the image processing community. Without effective testing, the performance of image processing algorithms such as red eye filters is questionable, and it can be hard to pinpoint problem areas in algorithms, or to assess algorithm accuracy. The next section explains what exactly a digital image processing algorithm is, and gives an overview of the history and evolution of the digital image processing field.

2.2.2 What is a Digital Image Processing Algorithm?

Central to the adoption of digital photography by both businesses and consumers have been advances in the areas of image processing and image analysis, particularly in the development of complex image processing algorithms. Digital image processing is a collection of techniques/algorithms for the manipulation of digital images by computers.

"*The field of digital image processing refers to processing digital images by means of a digital computer.*"(Gonzalez & Woods, 2002:1)

A digital image usually captured by a variety of input devices and techniques, such as digital cameras or scanners, is a picture that has been converted into a computer readable binary format represented as a matrix of elements called pixels (Gonzalez & Woods, 2002:2). The digital image contains instructions on how to colour each pixel which can be thought of as small dots on screen which together create the digital image.

"*Picture quality is strictly related to the number of pixels composing the sensor: the higher the better*" (Mancuso & Battiato, 2001:2)

Computer vision uses information extracted from such images in order to assist in decision making. Computer vision and image processing are related fields with computer vision using many of the techniques which traditionally belong to image processing. (Gonzalez & Woods, 2002:1) reports that there is no clear-cut boundary between the two. One formal distinction is that image processing deals with transforming images, producing one image from another, whereas computer vision deals with extracting specific information from images - for instance object recognition, the detection of known objects within an image, its main aim being to emulate human vision (Bowyer &

Phillips, 1998; Gonzalez & Woods, 2002:1; Meer et al, 2000). Both involve the analysis of digital images by computer algorithms.

"*Computer vision, for example, aims to duplicate human vision.*" (Low & Hjelmas, 2001:237)

An algorithm is a complex mathematical operation which can be defined as a procedure or formula for solving a problem. In relation to image processing the term algorithm is used to describe a problem-solving method suitable for implementation as a computer program. Standard features on most digital cameras sold nowadays, such as automatic red eye removal, are the result of complex mathematical operations known as algorithms. For instance the capturing of a photograph by a digital camera, a rudimentary task for most users, is driven by an elaborate processing sequence. Such tasks may seem simple but the underlying technology has been in development for the last 40 years.

2.2.3 Digital Image Processing History And Evolution

Ever since computers have become powerful enough to manage the processing of large data sets of images, researchers and developers have taken a great interest in image processing technology across the consumer and business landscape (Connolly, 2003:193).

"Computers, even PCs, are so fast and so well-endowed with storage that it is entirely feasible to process large datasets of images in a reasonable time — and this means it is possible to quantify the performance of an algorithm." (Clark & Clark, 2002:2)

(Clark & Clark, 2002:2) details the discipline variously known as computer vision, machine vision and image analysis as having its roots in the early artificial intelligence research of the late 1950's and early 1960's when digital computers first became available. (Umbaugh, 1998) describes image processing as originating during this period as an extension of electrical engineering and specifically digital signal processing.

"*The original goal of vision was to understand a single image of a scene, locate and identify objects, determine their structures, spatial arrangements, relationships with other objects, etc.*" (Shah, 2002:103)

The United States (US) government saw the potential of image processing in relation to defense, security and space exploration and during the 1970's and 1980's played a key role in establishing an infrastructure for the computer graphics field through support and research funding. During the 1960's, the National Aeronautics and Space Administration (NASA) converted from using analog transmission signals to digital signals with their space probes to map the surface of the moon (sending digital images back to earth). Computer technology was advancing at this time so NASA was able to use computers to enhance the images that the space probes were sending back. The digital and personal computer revolution of the 1980's and 1990's, spawned in part by the change from analog to digital, allowed major corporations in the private sector at the time (Bell Laboratories (http://www.bell-labs.com/, 2006) and General Electric Company (http://www.ge.com/, 2006)) to harness this new technology and develop products for the commercial market place (Jähne, 1997:32).

"*Rapid performance in computer technology and photonics had reached a critical level of performance*" (Jähne, 1997:32)

After the first digital camera for the consumer-level market that worked with a home computer was released by Apple Computer Inc. (http://www.apple.com, 2006) in February 1994 the digital camera market has since exploded and in tandem with the rapidly decreasing cost and increasing power of modern computers (Bovik, 2000:657), we have seen the creation of the multi-billion industry known as info-imaging. This multi-billion dollar industry involving 100's of new and established technology companies, which (Eastman Kodak, 2006) estimate as been worth $385 billion worldwide, combines three closely related markets: (1) Devices, (2) Infrastructure, (3) Services/media; which are all converging based on the central role of imaging. In effect the info-imaging industry is concerned with exploiting the latent information stored in personal and commercial images by using digital technology to extract the relevant data, and with creating new ways to capture, store and transmit information-rich images (Eastman Kodak, 2006). In essence images are information (Eastman Kodak info-

imaging, 2006), so image processing advances have not simply been confined to consumer software and electronics equipment market but have driven developments across many areas of computer vision.

The next section presents a selective review of some of the most popular image processing algorithms used today in both the consumer software and electronics equipment market and in the security industry.

2.3 WHAT IP ALGORITHMS MAKE POSSIBLE

2.3.1 IP ALGORITHMS IN USE IN CONSUMER DIGITAL IMAGERY

While first class artificial vision systems are still very much a concept in development terms, a series of highly evolved algorithms dominate the digital image market today. As (Bovik, 2000:243) reports, images are produced to record and display useful information but because of imperfections in the image capturing process, the recorded image invariably represents a degraded version of the original scene. So the correction of these imperfections to preserve image quality is critical to many of the ensuing image processing tasks (Bovik, 2000:243).

"*As the digital images are captured, stored, transmitted, and displayed in different devices, there is a need to maintain image quality.*" (Bovik, 2000:669)

Current Image Processing (IP) algorithms are powerful enough to distinguish subtle differences in image content and "pick up" only the elements of interest. Equally important is their ability to disregard the content that is particular to the environment in which the image was captured such as exposure or illumination. Much of this technology is already incorporated inside digital cameras as part of the image capturing process, or prior to display on personal digital assistants, mobile phones and other digital imaging appliances.

"*To support these uses, ease and speed are critical. Image quality needs to be 'good enough'*" (Van House et al, 2005:1856)

Without realising it, users of everyday digital consumer appliances such as digital cameras, software graphics packages and printers are harnessing years of research and development in image processing technology. IP algorithms embedded in digital cameras must perform a significant amount of data processing before the captured image can be compressed and converted to one of several image data formats, the most common being Joint Photographic Experts Group (JPEG) standard, a worldwide standard for the compression of digital images (http://www.jpeg.org/, 2006). Pre-capture IP algorithms are first applied to determine the three parameters which will determine the quality of the final picture: white balance, exposure and focus (Mancuso & Battiato, 2001:4). Following this, post capture IP algorithms may be applied to remove image defects and improve the quality of the images acquired. (Mancuso & Battiato, 2001:4; Messina et al, 2003:549).

"*Consistent image quality is one of the most important requirements for a camera system*" (Shirvaikar, 2004)

Among the most popular IP algorithms in use today are:

2.3.1.1 Red Eye Detection and Correction Algorithms

"Caused by light reflected off the subject's retina, red-eye is a troublesome problem in consumer photography" (Zhang (b) et al, 2004: 2363)

Red eye continues to be the most common customer complaint in the digital imaging market (Luo et al, 2004; Schettini et al, 2004:139; Smolka et al, 2003:1767; Zhang (b) et al 2004). The problem occurs when a flash is used to take a photograph and the light reflecting from human retina makes the eyes appear red in the photograph.

"*The objectionable phenomenon is well understood to be caused in part by a small angle between the flash of the camera and the lens of the camera.*" (Deluca Patent, 2002)

Most digital cameras sold today incorporate red-eye filters which analyze the captured image for the red-eye phenomenon and correct the image by changing the red area to its original colour. Powerful recognition software is able to distinguish between red eyes and other red dots within the photograph that may be of similar size and colour and correct the red eye discrepancy.

"*The digital camera has a red-eye filter which analyzes the stored image for the red-eye phenomenon and modifies the stored image to eliminate the red-eye phenomenon by changing the red area to black.*"(Deluca patent, 2002)

For the most part the corrected images are realistic and retain any highlights on the eyes. However the explosion in sales of camera phones means the problem has become more acute. With camera phones being small in size, the flash source is located even closer to the lens which results in a higher rate of red-eye pictures.

2.3.1.2 Dust Detection and Removal Algorithms

The appearance of dust on the plate on the front of digital camera lens is another common problem in digital photography. The dust is caused by small particles entering the camera body when changing lens. Attracted electrostatically, once on the sensor it can be very difficult to remove, resulting in impinged image quality. Aside from physically removing the dust particles from the image sensor the only other way to removes the defects in the image is to manually correct them. Current powerful dust detection algorithms can find the dust and reduce and eliminate its effect on the image by reconstructing the dust obscured area accurately.

2.3.1.3 Auto Exposure Algorithms

One side effect of the digital imaging revolution is that digital cameras perform differently than film cameras in their treatment of highlights and shadows. The exposure parameter is the amount of light that hits the image-sensor and determines how light or dark the captured image will be (Mancuso & Battiato, 2001:4). Controlling exposure is crucial to capturing the desired image. If too much light gets through to the image sensor

the resulting image will be over exposed and have a faded-out look whereas if too little light gets through to the sensor the resulting image will be underexposed and have a darkened look. (Messina et al, 2003:549) report that this is a difficult problem to solve particularly in mobile camera phones where several factors including absence of flash gun contribute to badly exposed images. Most high-end digital cameras sold nowadays incorporate histogram tools used to filter the amount of light hitting the sensor, resulting in clearly exposed images. A histogram is a graph of the distribution of brightness values of individual pixels in a digital image. The left hand side of the histogram indicates the dark tonal range referred to as shadows, the centre portion indicates the mid-tones of the image, while the right hand side of the histogram shows the bright tonal range or highlights of the image.

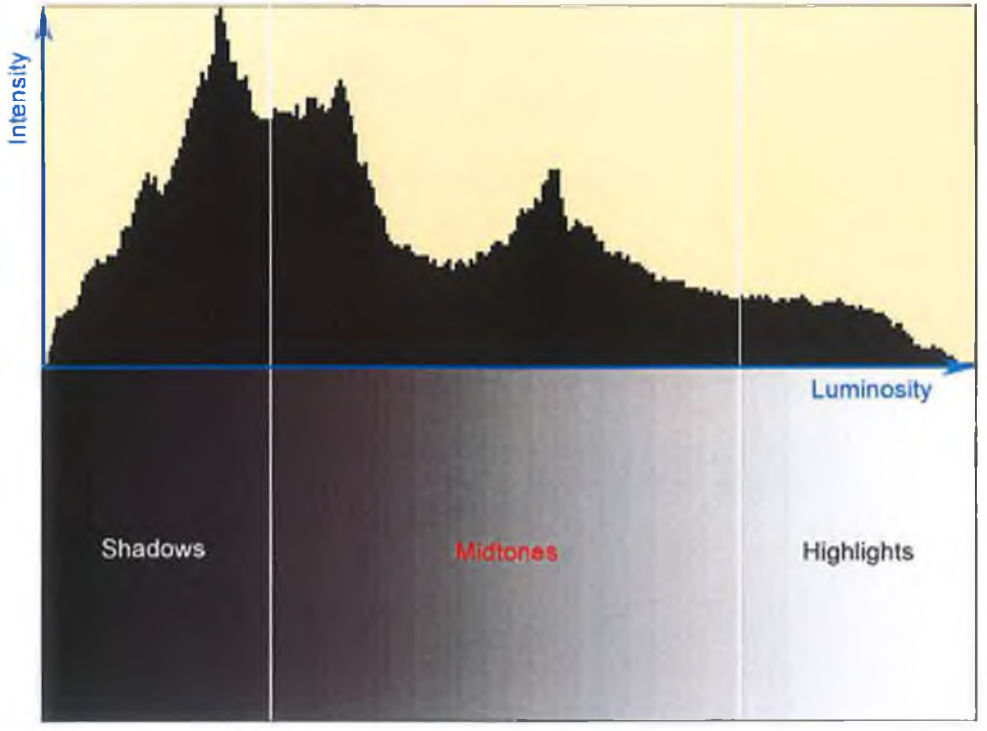


Figure 2: Image Histogram

As illustrated in the Figure 2 above, if the left hand side or right hand side of the histogram is empty the image will be most likely be over or under exposed.

Unfortunately, according to (Messina et al, 2003:549), there is not an exact definition of what correct exposure should be but developers are all the time working on new auto-exposure algorithms to capture clearly exposed images.

The beauty of current IP algorithms like those mentioned above is that they can be optimised to run inside digital cameras, printers and other devices with relatively low on-board processing power. This means the process is completely automatic and transparent in operation and relieves much of the disappointment and stress of those who use digital cameras. Developers are all the time improving algorithms to make the process efficient in terms of computing resources but still fast enough on low-end embedded appliances to be practically unnoticeable to end-users. Of course as consumers yearn for sharper, brighter and clearer images, a lot of IP algorithm development and refinement still needs to be carried out (Riopka & Boult, 2003). The quality of an algorithm can have a strong impact on the perceived quality of the image. Ideally algorithms are designed to maximise classification accuracy whilst minimising computational effort.

"*Simpler algorithms translate into less hardware, lower power consumption, and lower cost*" (Mancuso & Battiato, 2001:1)

The rapid transition to digital photography has raised the expectations of non-professional photographers, and for the digital imaging research community the ultimate goal is to allow consumers capture and manage discrepancy-free images quickly and easily.

2.3.2 Computer Vision: IP Algorithms in Use in the Security Industry

Of course the use of IP algorithms has not been confined to individual consumer appliances like digital cameras. The security industry has for the last 20 years embraced computer vision technology; incorporating algorithms that deduce information from visual images, including face recognition and fingerprint analysis technologies (Hong et al, 2004:103; Kung et al, 2004).

"*Image is better than any other information form for our human being to perceive. Vision allows humans to perceive and understand the world surrounding us.*"
(http://iria.math.pku.edu.cn/~jiangm/courses/dip/html/node3.html, 2003)

From a security viewpoint the need to duplicate human vision has been one of the main driving forces behind developments in image processing. (Gur, 2002) proposes that the technology holds the possibility of developing the ultimate machine in the future that will be able to perform the visual functions of human beings. Understandably since September 11 (http://www.september11news.com/, 2005) in New York and July 7 (http://news.bbc.co.uk/1/hi/in_depth/uk/2005/london_explosions/default.stm, 2005) in London, increasing international attention has been brought to imaging technology as security concerns have grown worldwide (Kung et al, 2004; Mazor, 2005).

The greater interest in security has led to biometrics, the ability of a computer to recognise a human through a unique physical trait, becoming one of the fastest growing fields in advanced technology. (Hong et al, 2004; Kong et al, 2005:104; Pankanti et al, 2000; Prabhakar et al, 2003:33) report that biometrics systems have the best performance in terms of security, management and user convenience compared to traditional systems able to perform the visual functions of human eye. Traditionally biometric authentication has been used for the purpose of either verification or identification in law enforcement to identify criminals (Jain et al, 2004:19). Nowadays it is increasingly being used to identify persons in a large number of civilian, commercial and financial applications (Bovik, 2000:821; Hong et al, 2004:104; Pentland & Choudhury, 2000:54; Yang et al, 2004). As biometric imaging comes in many varieties and exists to meet a broad set of needs, (Jain et al, 2004:10; Prabhakar et al, 2003:36) have divided biometric applications into three main groups:

(1.) **Commercial Applications** e.g. computer network login, electronic data security, e-commerce, physical access control, etc.

(2.) **Government Applications** e.g. border and passport control, national ID card, drivers license, etc.

(3.) **Forensic Applications** e.g. terrorist identification, criminal investigation, parenthood determination, etc.

Perhaps unbeknownst to us as consumers the above applications have become a part of everyday life in the 21st century.

"But it is certain that biometric-based recognition will have a profound influence on the way we conduct our daily business."(Jain et al, 2004:19)

As the prospect of a biometric imaging industry explosion looms large one biometric that will garner a lot of attention will be face recognition (Kim et al, 2004; Pentland & Choudhury, 2000). Face recognition involves:

"*Identifying or verifying one or more persons of interest in a scene by comparing input images with face images stored in a database.*"(Kong et al, 2005:104)

Evidence of face's recognitions importance can be seen in the wide array of research carried out in the field and the wide variety of face detection and recognition techniques which have been proposed (Kim et al, 2004;Kung et al, 2004; Little et al, 2005:89; Pentland & Choudhury, 2000:51; Yang et al, 2004:2533). Different face recognition systems use different methods of facial recognition; however all focus on measures of key features of the face. The process usually involves the scanning of a person's face and matching it against a library of known faces for a match. Usually face detection or segmentation is first carried out, then feature extraction, followed by the recognition process which usually involves either identification or verification (Kong et al, 2005:104; Zhao et al, 2003: 400).

"Given an arbitrary image, the goal of face detection is to determine whether or not there are any faces in the image and, if present, return the image location and extent of each face." (Yang et al, 2002:34)

Since a person's face can be captured by a camera from a distance away, facial recognition has a covert or concealed operation (i.e. the subject does not necessarily

know he has been observed) (Kong et al, 2005:127; Kung et al, 2004; Pentland & Choudhury,2000:50; Prabhakar et al, 2003:41). For this reason, the demand for this technology will probably increase further as both government and private organisations search for more effective military surveillance, monitoring and image recognition technologies for surveillance and biometric identification tasks (Kung et al, 2004; Yang et al, 2004). (Prabhakar et al, 2003:42) conclude in 2003 that it is too early to predict where and how biometrics will evolve, but as reliable personal recognition has become critical to many business processes, what is certain is that biometric-based recognition will profoundly effect the way we perform our daily business. For these applications to maintain a very high degree of security there is still plenty of scope for improvement in this area.

"*Although there are numerous algorithms today that can achieve acceptable recognition rates on idealized image set, there exists no algorithm capable of adequately recognizing people in real-world situations*" (Little et al, 2005:89)

(Torres, 2004:21) reports no technique exists that provide robust solutions to all the situations a face recognition system may encounter. And with (Balasuriya & Kodikara 2001) reporting that automated face recognition has become the holy grail of computer vision artificial intelligence, the question arises: how do we test and compare algorithm performance? Currently no standard methodology exists to guide algorithm development (Black et al, 2002; Liu & Dori, 1999; Sharma & Reilly, 2003; Thacker et al, 2003).

"*Although the evaluation and validation of algorithms have been discussed for over a decade, the research community still faces a lack of well-defined and standardized methodology.*" (Takeuchi et al, 2003:408)

As a result, algorithms are often implemented based on programmer's intuition and experience. It is only extensive testing that proves an image processing algorithm's accuracy. The next section will examine some of the major challenges faced by algorithm developers in producing completely robust IP algorithms particularly face detection and recognition and red eye detection and correction algorithms and explain how an

algorithm performance measurement framework would greatly assist the algorithm development process.

2.4 ONGOING CHALLENGES IN IP ALGORITHM DEVELOPMENT

Current IP techniques and algorithms may be resolving many common picture defects in consumer photography and making the world a more secure place but there still are numerous challenges facing the image processing industry. (Delac et al, 2005:136) reports in 2005 that the Pose, Illumination and Expression variant (PIE) problem is still the most studied issue in face recognition so far while (Kong et al, 2005:105) reports that evidence of the depth of the illumination problem can be gauged in that variations of face images due to illumination variation and viewing direction are typically larger than variations raised from changes in face identity. Similarly problems relating to illumination, camera lens and photographic subject mean no current red eye detection and correction IP algorithm is completely effective (Smolka et al, 2003; Zhang, 2004b). The next section explains some of the main challenges in producing truly effective face detection and recognition algorithms and red eye detection and correction algorithms.

"*It is my personal view that computer vision is a hard problem*" (Shah, 2002:103)

2.4.1 FACE DETECTION AND RECOGNITION ALGORITHMS

In face recognition where greater accuracy is the key pursuit for facial recognition systems, performance has been mixed (Gross et al, 2001; http://www.biometricgroup.com/, 2006). In 2004, (Gross et al, 2004) reported that the most recent evaluation of commercial face recognition systems shows the level of performance for face verification of the best systems to be on par with fingerprint recognisers for frontal, uniformly illuminated faces. Under limited conditions, for example, verification under controlled indoor environments, performance is quite good.

These developments have meant face recognition has become a key component in many biometric systems (Kung et al, 2004).

However, although face recognition has reached a significant level it is still far away from the capability of human perception (Dalong et al, 2005; Little et al, 2005;Kung et al, 2004; Torres, 2004; Zhao et al, 2000). Despite over 30 years of research in the area, (Dalong et al, 2005:787) reports that robust recognition of faces in digital photographs, especially family photographs, still remains a challenge. The US government-sponsored Face Recognition Vendor Test (FRVT) (Phillips et al, 2003) to measure progress on difficult face recognition problems in 2002 found that in the images taken indoors, where environmental conditions can be better controlled, the current state of the art in face recognition is 90% verification at a 1% false accept rate. Nevertheless, FRVT (Phillips et al, 2003) found that face recognition with variant pose, illumination and expression (PIE) is still far from adequate (Dalong et al, 2005:787).

"*Complex visual events arise for which robust interpretation requires separating the external causes from the intrinsic properties in the appearance of each object. This task, roughly equivalent to perceptual constancies in human visual perception, is currently an active research area in computer vision*" (Meer et al, 2000:2)

Since the early 1990's many papers (Adini et al, 1997; Gross et al, 2004; Kim et al, 2004; Little et al, 2005; Lu, 2003; Pentland & Choudhury, 2000:51; Riopka & Boult, 2003; Torres, 2004; Yang et al, 2004; Zhao et al, 2000; Zhao et al, 2002) have been written identifying and proposing solutions to what has become known as the PIE (Pose, Illumination, Expression variant) problem (Dalong et al, 2005:787). (Delac et al, 2005:136) notes up to 2005 that the PIE problem is still the most studied issue in face recognition so far. According to (Dalong et al, 2005:793; Kim et al, 2003:29; Kim et al, 2004, Yang et al, 2004) the primary source of difficulty in developing robust face recognition systems is pose variation where the same face appears differently due to changes in viewing conditions. Similarly the illumination variation problem involving the

same faces appearing differently due to changes in lighting (Yang et al, 2004; Zhao et al, 2002:379) is another key barrier.

"Meanwhile, current automatic face recognition technology is not sufficiently robust to consistently identify people in widely varying lighting conditions and poses." (Girgensohn et al, 2004a:99).

Evidence of the depth of the illumination problem can be gauged, according to (Kong et al, 2005:105), in that variations of face images due to illumination variation and viewing direction are typically larger than variations raised from changes in face identity.

"While humans quickly and easily recognize faces under variable situations or even after several years of separation the problem of machine face recognition is still a highly challenging task in pattern recognition and computer vision." (Kong et al, 2005:104)

According to (Bovik, 2000:837) the development of recognition and detection systems for natural objects such as human face is difficult because they are complex, multidimensional, and important visual stimuli. Since (Bowyer & Phillips, 1998) commented in 1998 on the common belief that computer vision was poised to deliver reliable solutions, the current status of face recognition technology has certainly advanced. But in terms of performance, current face recognition technologies are still far from that of human vision. According to (Heseltine et al, 2003:59) because of the difficulties like the PIE problem leading to high error rates, face recognition technology has yet to be put to widespread use in commerce or industry. In the area of face recognition (Torres, 2004) points out

"In spite of the great work done in the last 30 years, we can be sure that the face recognition research community will have work to do during, at least, the next 30 years to completely solve the problem." (Torres, 2004:3).

2.4.2 Red Eye Detection and Correction Algorithms

Similarly problems relating to illumination, camera lens and photographic subject mean no current red eye detection and correction algorithm is completely effective (Smolka et al, 2003; Zhang, 2004b). According to (Schettini et al, 2004) the main challenge in effective red eye correction is to avoid the correction of what are called false positives,

i.e. misclassified red spots that can be found in the image, while maintaining high correction rates and quality. Some algorithms identify red eye pixels too aggressively, darkening eye lid areas, while others are too conservative, leaving many red eye pixels uncorrected.

"It is generally hard for an automatic detection algorithm based on statistical techniques to handle all red-eye cases" (Zhang, 2004b:2365).

And a lot of the automatic red eye solutions that do exist are heavily dependent on face detection algorithms to detect the red eye regions which may in turn suffer from challenges in face detection technology (Luo et al, 2004; Zhang, 2004b). Due to these problems many image processing applications - "IPhoto" (http://www.apple.com, 2006), "Picture Maker" (Eastman Kodak, 2006) - that offer red eye solutions on the market today are semi-automatic or manual solutions (Luo et al, 2004). Essentially because certain algorithms are not accurate enough for automatic detection some systems must combine user input in semi-automated detection systems (Girgensohn et al, 2004a; Schettini et al, 2004; Smolka et al, 2003; Zhang et al, 2003)

What is clear is image processing algorithm development is not easy, particularly object detection. Contrast current face recognition algorithms having a better performance on photographs captured indoors while (Zhang et al, 2002:99) find in the area of image orientation detection that after analysing image orientation detection results in detail they report that the accuracy of indoor images is much lower than that of outdoor images.

"*Humans identify the correct orientation of an image through the contextual information or object recognition, which is difficult to achieve with present computer vision technologies*". (Zhang et al, 2002:95)

Although a number of efforts have been made to solve the difficulties outlined above, the performance of current algorithms are still not satisfactory (Lu, 2003; Müller et al, 2004; Sharma & Reilly, 2003; Torres, 2004; Zhang et al, 2003). With new approaches being developed to overcome the challenges in robust algorithm development mentioned above,

the question of how we test and compare algorithm performance becomes more critical. Furthermore where algorithms have to be developed and validated quickly as in the field of face recognition, security and protection applications, it is of crucial importance that we are able to empirically study their performance (Jaynes et al, 2005:1). The next section will define what is meant by performance evaluation from a digital image processing algorithm perspective, highlight its need and determine some of the problems hindering its effective implementation in the image processing area.

2.5 IP ALGORITHM PERFORMANCE MEASUREMENT

Proper evaluation has always been important in any research area. Only with proper and standardised evaluations can advances in the field be identified and promoted. Unfortunately the field of image processing currently lacks a testing framework, for developing and assessing the performance of IP algorithms. And with algorithms being developed without their likely performance being calculated beforehand, this often leads to a somewhat ad hoc approach to development. Apart from the area of biometrics and face recognition in particular, evaluation methodologies, standards and protocols are still largely unavailable. The next section will define what is meant by performance measurement from a digital imaging algorithm perspective, highlight its need and determine some of the problems hindering its effective implementation in the field of image processing.

"*The application of any technology should be based on the careful consideration of sound scientific test results*" (Bone & Backburn 2002:7)

2.5.1 NEED FOR IP ALGORITHM PERFORMANCE MEASUREMENT?

According to (Blackburn, 2001) successful evaluation allows the strengths and weaknesses of a technology to be shown so we understand where it can be deployed and what areas future development efforts should be focused on. Benchmarking and

performance evaluation not only allows the comparisons of different IP algorithms and appraisal of the best suited IP algorithm for a particular problem but they also allow algorithm developers to identify performance bottlenecks and design better systems (Pankanti et al, 2000). By enabling researchers to measure the performance of their IP algorithm relative to existing techniques, a developer can quantify any improvements that each new algorithm development iteration causes, and so speed up the algorithm development process. Furthermore if system performance for a given task can be predicted from previously acquired data, its suitability for the task can be evaluated faster thereby reducing development time. And knowledge of the performance of different IP algorithms aids the selection of the most appropriate technique for a given problem.

2.5.2 Historical Lack of Research into IP Algorithm Performance Measurement

Historically the lack of any standard performance evaluation of IP algorithms has been one of the main problems hindering development across all areas of image processing.

> *"As vision techniques become more widely applied, the need to critically evaluate new methods has also become recognised by users."* (Courtney & Thacker, 2003:2)

Essentially over the last 50 years a large variety of image processing techniques/algorithms for processing image data have been developed but according to (Chhabra & Phillips, 1998; Phillips et al, 1998a; Takeuchi et al, 2003) very little work has been done in the area of measuring and analysing the performance of these algorithms. In the early 1990's (Firschein et al, 1993) outlined the need for benchmarking in the vision field to objectively measure its progress. (Firschein et al, 1993) pointed out that, similar to advancements in the natural language and speech recognition fields, rigorous comparison among various computer vision techniques would help to spur advancements across the image processing algorithm development industry. In 1997 (Jähne, 1997:49) suggested that a better mathematical foundation would lead to image

processing algorithms becoming more predictable and accurate paving the way towards faster and more efficient algorithms in the long run.

"An important issue is also that a detailed mathematical analysis also leads to faster and more efficient algorithms, i.e. it becomes possible to perform the same image processing task with less operations" (Jähne,1997:49).

And in 1998 (Bowyer & Phillips, 1998) reported that a better comparative assessment of algorithms would have the following benefits:

(1) Place computer vision on a solid experimental and scientific ground.

(2) Assist in developing engineering solutions to practical problems.

(3) Allow accurate assessment of the state of the art.

(4) Provide convincing evidence to potential users that computer vision research has indeed found a practical solution to their problems.

Up to the beginning of the 21st century (Meer et al, 2000:6) made the point that a better mathematical foundation of image processing was not just crucial to the area of image processing performance evaluation but would also be of great benefit in algorithm design and development, similar to arguments made by (Jähne, 1997:49; Micheals & Boult, 2000) years previously. At this time however according to (Bowyer & Phillips, 1998) the computer vision community failed to heed these arguments. (Chhabra & Phillips, 1998; Liu & Dov, 1999; Ojala et al, 2002) were to report in the late 1990's that the performance of most techniques were at best only known from the biased and subjective reports of the algorithm developers. And up to 2001 (Courtney & Thacker, 2001:1) reported that

"Some 30 years of research has produced a rich variety of methods for processing image data, but little information on how they perform beyond a few example images."

2.5.3 Reasons for Lack of Research into IP Algorithm Performance Measurement

"Although the evaluation and validation of algorithms have been discussed for over a decade, the research community still faces a lack of well-defined and standardized methodology." (Takeuchi et al, 2003:408)

There are a number of reasons why performance evaluation has not been commonly practiced in the image processing community. In 2003 (Thacker et al, 2003:3)details two reasons why the need for algorithm characterisation has not been emphasised; one being a poor understanding and use of statistics and two being the lack of knowledge on how to use the generated data. (Courtney & Thacker, 2001:3; Micheals & Boult, 2001:150) is of the opinion that the failings in algorithmic reliability have been due to the neglect of the important role that statistics must play in algorithm development. Performance evaluation is not just finding out whether algorithms perform as expected; according to (Courtney & Thacker, 2001), it involves the use of objective, usually statistical, measures for comparing the performance of vision algorithms. (Courtney & Thacker, 2001; Micheals & Boult, 2000). Since the overall performance of an IP algorithm is a function of both the effectiveness of the algorithm and the conditions under which it operates, the decoupling of these two factors can be difficult. Describing complex image conditions quantitatively can be hard. Different tasks require different performance measures; the metrics for characterising a detection algorithm will differ from that of an optical character recognition algorithm (Föerstner, 1996).

"*The variety of tasks leads to a variety of requirements*" (Föerstner, 1996:3)

Furthermore vision is complex and many IP algorithms are developed without accompanying reference material making performance difficult to analyse. Finally because testing is time-consuming and the gathering of ground truth can be expensive and dubious to acquire, historically researchers paid little attention to IP algorithm performance evaluation. It is only in the last 10 years that IP algorithm performance evaluation has become recognised as a valuable research area.

2.5.4 Recent Interest in IP Algorithm Performance Measurement

"However, it has become well accepted that performance evaluation is a critical component in validating existing and new algorithms." (Micheals & Boult, 2001:150)

According to both (Courtney & Thacker, 2001; Hua et al, 2004:498; Meer et al, 2000:2 Micheals & Boult, 2001:150) since the start of the 21st century the machine vision community have took on board that a more rigorous approach to the studying of performance characterisation of vision algorithms was needed.

"*Performance evaluation and benchmarking have been gaining acceptance in all areas of computer vision*" **(Phillips & Chhabra, 1999:1)**

Unquestionably since (Chhabra & Phillips, 1998; Phillips et al, 1998) reported in 1998, performance evaluation as still a very young field, a more concerted effort by developers and researchers to better analyse performance has taken place. Both (Bowyer & Phillips, 1998; Takeuchi et al, 2003:408) report on the increased publications addressing the issue of how to evaluate the performance of vision algorithms while in 2005 (Grgic et al, 2005) reported in the face recognition area that for each newly developed algorithm at least one paper is written comparing that algorithm to other more known algorithms.

"*As a result, the vision community has finally started to turn its attention to issues related to testing and comparing algorithms: performance assessment.*" (Clark & Clark, 2002:2)

(Clark & Clark, 2002) identifies rapid developments in Personal Computer technology making it possible to quantify the performance of an algorithm as being one of the factors behind this movement. As a result a number of standard techniques are available for measuring the performance of IP algorithms. These include performance characterisation, performance evaluation, benchmarking and standardisation of image databases. The next section provides an overview of each technique.

2.5.5 Techniques available for IP Algorithm Performance Measurement

"*Its principal aim is to provide information, datasets and software that allow the effectiveness of algorithms to be measured and compared. This is known variously as performance characterization, performance estimation and benchmarking.*" (http://peipa.essex.ac.uk/, 2006)

2.5.5.1 Performance Characterisation

(Thacker et al, 2005:5) define performance characterisation as referring to specifically obtaining a sufficiently quantitative understanding of performance that the output data from an algorithm can be interpreted correctly. It involves the measurement or calculation of the performance of an algorithm throughout the full space of the expected operating conditions. Different methods for performance characterisation have been developed, such as testing. Testing simply involves implementing the system and testing it on its real data. Unfortunately testing can be time consuming because the IP algorithm must be implemented and a set of input data with the appropriate variation in operating conditions must be acquired. Furthermore (Ojala et al, 2002:705) reports often the requirements for testing have more to do with the complexity of describing the operating conditions than with the IP algorithm itself.

2.5.5.2 Performance Evaluation

Performance evaluation differs from performance characterisation in that it measures performance under the conditions in which the algorithm will be operating in. Certain performance characteristics, which it would be necessary to measure for complete characterisation, can then be ignored. Therefore it is easier to implement performance evaluation although it may not give a complete description of system performance.

(Takeuchi et al, 2003:408) classifies performance evaluation approaches into the following general categories:

> ***"Comparative:*** *Here an algorithm may be compared with others that attempt to address the same image processing task, or its performance may be compared to "ground truth," or perhaps to human performance.*
>
> ***Analytic:*** *The theory behind the algorithm is examined to try to determine the limits to its operation. The computational complexity may be derived, or theoretical optimality may be determined under certain constraints. Frequently, the approach makes use of simplified input data to make the analysis feasible.*
>
> ***Performance:*** *The way the algorithm actually performs on test data is measured and execution times with different parameters may be reported.*

Appropriateness to Task: *The algorithm is shown in the context of a particular application, and the constraints of the task are used to justify the selection of the particular algorithm. The performance of the task as a whole is taken as the evaluation of the algorithm."* (Takeuchi et al, 2003:408)

More informal measures including generality and acceptance are also mentioned. However, the lack of effective algorithm performance evaluation up to now is evident as noted by (Takeuchi et al, 2003:409). They find that perhaps the only real performance evaluation measure in common use is longevity i.e. the best IP algorithms are those that are accepted widely and implemented by many people for different applications.

2.5.5.3 Benchmarking

Benchmarking differs substantially from the previous two techniques, a benchmark being a basis to compare performance. (Loy & Eklundh, 2005) defines the aim of a benchmark as not just to evaluate the performance of an individual IP algorithm, but to allow direct comparison of different IP algorithms. (Clark & Clark, 2002) uses performance characterisation and defines it.

"one must explore what characteristics of the inputs affect the algorithms' performances and by how much." (Clark & Clark, 2002:3)

(Firschein et al, 1993; Loy & Eklundh, 2005) suggests benchmarking's main motivation is to provide a basis for comparing different IP algorithms, and to track progress towards human-level performance. In essence it allows researchers to more fully understand the strengths and weaknesses of their IP algorithm and compare their results with other IP algorithms.

2.5.5.4 Standardisation of Image Databases

Several other techniques have been developed which are related to the measurement of IP algorithm performance. One such approach to comparing IP algorithm performance is to provide databases of standard images, covering a wide range of operating conditions against which algorithms can be tested. As (Yang et al, 2002:50) point out the understood reason for comparing algorithms on test sets is that these datasets represent problems that algorithms will encounter in the real world and that superior performance on these

datasets should in turn translate to superior performance on other real world tasks (Phillips et al, 2005).

Therefore as complex IP algorithms begin to evolve; each new version of an image processing algorithm should be verified against a large set of reference images to ensure that an actual performance improvement has occurred. Since performance characterisation of all the operating conditions for a particular algorithm is next to impossible (Moon et al, 2002:1) in most cases it is adequate to test an IP algorithm under the expected conditions it will be operating in (Jaynes et al, 2005:2). As mentioned in section 2.5.5.2 this technique is known as performance evaluation. The next section provides an in-depth explanation of the performance evaluation of IP algorithms, it being the most practical means of evaluating algorithm performance.

2.5.6 Performance Evaluation

"Statistical measures of performance can be obtained by testing on a representative set of data." (Thacker et al, 2003:5)

Performance evaluation of IP algorithms is a black box evaluation methodology (Thacker et al, 2005:8), whereby the internal workings of an algorithm are not investigated; rather the output is compared with the expected results. The reasoning behind the use of black box evaluation according to (Ojala et al, 2002:704) is given a task, what really matters in terms of performance is the quality of the output and the (computational) cost, not the internal properties of the algorithm.

"The task of a computer vision algorithm can be specified in terms of two components: the range of images to be processed and the performance criterion that the algorithm should try to achieve." (Moon et al, 2002:1)

Evaluating IP algorithm performance in this way involves the creation and running of test scenarios incorporating the algorithm on sample data sets to deduce performance scores (Clark & Clark, 2002:15; Courtney & Thacker, 2001). A scenario or application evaluation focuses on understanding the performance of specific IP algorithm designed to

do a specific task (Courtney & Thacker, 2001:3; Phillips (a) et al, 2000; Thacker et al, 2003:3; Thacker et al, 2005:6). The aim of the scenario evaluation is to determine the underlying technical ability of a particular technology (Courtney & Thacker, 2001:3; Phillips (a) et al, 2000; Thacker et al, 2003:3; Thacker et al, 2005:6). Results from the evaluations usually presented in terms of output parameters or performance data then show specific areas of the algorithm that require future research and development. To perform evaluations in this way (Clark & Clark, 2002:4) reports that each individual test scenario requires three pieces of information:

(1.) The **Test Data** is the actual input to the algorithm under test. Common practice is to divide test data into a small number of cases and qualitatively compare performance across cases (Thacker et al, 2005:28) The original FERET (Phillips et al, 2000b) tests for example compared performance under three qualitatively stated conditions: 1) same day images with different facial expressions, 2) images taken under changing illumination, and 3) extended lapsed time between acquisition of images (Kong et al, 2005:111; Thacker et al, 2003:28).

(2.) The **Ground Truth** is the corresponding expected output from the algorithm determined by a human.

(3.) Whether the output corresponds to a success or failure. The ground truth is compared with the actual output from the test run using **Metrics of Interest** to evaluate algorithm performance (Liu & Dori, 1999:98)

The results of the tests are statistically analysed to deduce information on algorithm performance. According to (Hua et al, 2004:498, Liu & Dori, 1999) metrics of interest are usually employed to express the difference between the expected output (ground truth) and the actual output (returned results) from the algorithm test. For a detection algorithm the returned results from the test usually take the form of four quantities (Clark & Clark, 2002:3; Liu & Dori, 1999; Sharma & Reilly, 2003):

True Positive: (Also known as detection rate, recognition rate, true acceptance or true match) occurs when a test that should yield a correct result does so.

True Negative: (also known as true rejection or true non-match) occurs when a test that should yield an incorrect result does so.

False Negative: (also known as false rejection, false non-match) occurs when a test that should yield a correct result actually yields an incorrect one.

False Positive: (also known as false acceptance, false match, false alarm) occurs when a test that should yield an incorrect result actually yields a correct one.

The evaluation process involves keeping track of the four above mentioned values – test metrics. According to (Yang et al, 2002:35) in the area of face detection most papers compare the performance of algorithms in terms of detection and false alarm rates. (Sharma & Reilly, 2003; Yang et al, 2002:35) defines detection rate as the ratio between the number of regions correctly detected and the number of regions that are determined by a human, the ground truth. Similar to detection rate, according to (Liu & Dori, 1999:103) the performance of recognition algorithms is usually reflected by the two rates, true positive and false positive, true positive or recognition rate being the ratio of the number of correctly recognised features to the total number of ground truths and false positive rate being the ratio of the number of incorrectly recognised features to the total number of recognised features (Liu & Dori, 1999:103). The algorithm with the highest detection or recognition rate (or equivalently, the lowest false rate) is normally seen in comparisons and competitions as the best.

Accordingly it must be appreciated that there is always a trade-off between true positive and false positive detection (Clark & Clark, 2002:4; Liu & Dori, 1999:103; Martin et al, 1997). If detection rules are too detailed the algorithm may fail to detect regions that do not pass rules whereas if rules are too general algorithm may return false positives.

"*In general, detectors can make two types of errors: false negatives in which faces are missed resulting in low detection rates and false positives in which an image region is declared to be face, but it is not.*" (Yang et al, 2002:35)

Similarly according to (Mu et al, 2001: 2877) designing a good face recognition system involves solving the two types of recognition problems simultaneously:

1. **True positive:** Correctly recognise and identify individuals who are in the database.
2. **False positive:** Reject images of individuals who are not part of the database.

(Mu et al, 2001: 2877) reports that a lot of face recognition algorithm designs are trained and tuned to work well on only one of these face recognition problems. As (Clark & Clark, 2002) explains:

"If a procedure is set to detect all the true positive cases then it will also tend to give a larger number of false positives. Conversely if the procedure is set to minimize false positive detection then the number of true positives it detects will be greatly reduced." (Clark & Clark, 2002:5)

Consequently for a fair evaluation developers should always appreciate that for detection or recognition algorithms there is always a trade off between true and false detection. (Courtney & Thacker, 2001:5; Yang et al, 2002:35). Furthermore, as (Clark & Clark, 2002:10) highlight, fair evaluations must not only take into account the number of true positive or false positives etc but also the size of test sets applied to the algorithm.

"This must take into account not only the number of false positives etc. but also the number of tests: if one algorithm obtains 50 more false positives than another in 100,000 tests, the difference is not likely to be significant; but the same difference in 100 tests almost certainly is." (Clark & Clark, 2002:10)

Unfortunately a lot of problems are evident in the literature as regards the performance evaluation framework mentioned above, among them the lack of a common methodology for the performance evaluation of detection and recognition algorithms as identified by (Clark & Clark, 2002; Heo et al, 2003:558; Liu & Dori, 1999:98; Sharma & Reilly, 2003). And with the greater interest by the image processing algorithm research and development community in studying algorithm performance the need for a comprehensive performance evaluation tool for all types of image processing algorithms becomes more important. According to (Courtney & Thacker, 2001), biometrics has been the first field of image processing to try and identify best practice in the area of

performance evaluation and (Kong et al, 2005:128; Phillips et al, 2000a) report that evaluations have already enhanced biometric performance.

The next section examines current performance evaluation techniques in the field of biometrics and highlights how areas of best practice there can be applied across all image processing algorithm evaluations.

2.5.7 A Successful Performance Evaluation Framework - Biometrics

With growing security concerns in recent times, the performance of biometric applications has become a high priority. So the evaluation methods and techniques for biometric algorithms have had to keep up to date so that meticulous testing can be conducted, helping algorithms to evolve and improve at a fast rate. According to (Phillips et al, 2000a)

> *"Evaluations in general—and technology evaluations in particular—have been instrumental in advancing biometric technology. By continuously raising the performance bar, evaluations encourage progress."* (Phillips et al, 2000a:62)

Major evaluations like the FERET (Face Recognition Technology) (Heo et al, 2003; Phillips et al, 2000b) and FRVT (Face Recognition Vendor Test) (Phillips et al, 2003) tests, which evaluated emerging approaches to face recognition (Pentland & Choudhury, 2000:52), have helped to measure the power of face recognition technology and have also served to drive its evolution. The FERET evaluation protocol provided a means for evaluating the performance of face recognition algorithms (Kong et al, 2005:111). The FERET program sponsored by the Department of Defence's Counterdrug Technology Development Program from 1993 through 1997 has encouraged and measured improvements in face recognition systems performance by providing a large database of facial images and a testing procedure to evaluate face recognition algorithms, two of the critical requirements in support of producing reliable face-recognition systems (Phillips et al, 2000b).

The earliest FERET Database test in August 1994 established for the first time a baseline for face recognition algorithms. The test was designed to measure performance on algorithms that could automatically locate, normalise, and identify faces from a database. Further tests have been carried out since to measure progress and evaluate algorithms on larger galleries with the Facial Recognition Vendor Test 2000 (FRVT 2000) (Blackburn et al, 2001) releasing its results in February 2001. In March 2003 the Face Recognition Vendor Test 2002 completed the most complete evaluation to date of commercially available face recognition systems finding that given reasonable controlled indoor lighting, the current state of the art in face recognition is 90% verification at a 1% false accept rate. Further tests are currently being carried out with the FRVT 2006 scheduled to run in January 2006.

While FERET and FRVT evaluations have certainly helped to assess the effectiveness of facial recognition technology, problems are still evident in its datasets. (Thacker et al, 2005:28) reports that the FERET database is poor in terms of the number of replicate images per subject it stores and (Black et al, 2002:6; Little et al, 2005:89) finds that the FERET database does not possess a wide enough variety of illumination or pose variations or information on the lighting used to capture the images.

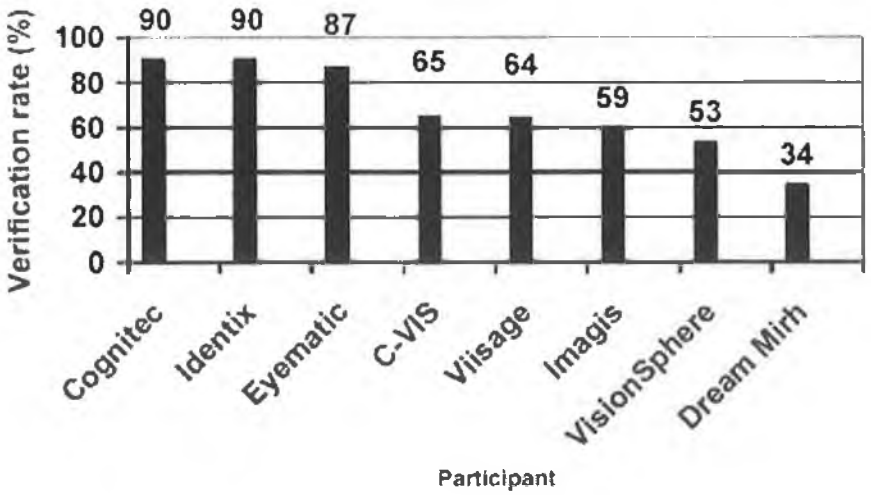


Figure 3: Sample results from the Face Recognition Vendor Test 2002

The results of the Face Recognition Vendor Test 2002 illustrated in Figure 3 above, found that FaceIt, a face recognition software solution, had the highest accuracy of any commercial facial recognition software (Heo et al, 2003:551; Kim et al, 2004:1482; Kong et al, 2005:112; Pentland & Choudhury, 2000:52). Its manufacturer Identix reported at the time (Bone & Backburn, 2002) that its results demonstrated that facial recognition is not only effective as a mechanism to prevent unauthorised access, but also as a tool to detect the presence of a criminal or terrorist attempting to access a restricted area. However contrary to (Bone & Backburn, 2002) findings, (Heo et al, 2003; Kim et al, 2003:36; Kim et al,2004; Riopka & Boult, 2003) find that although face-recognition systems such as FaceIt works well with "in-lab" databases and ideal conditions, up to 2004 none of the face recognition systems tested in airports have spotted a single person actually wanted by authorities. Furthermore (Heo et al, 2003:558) points out the technology has only served to embarrass innocent people by making incorrect matches, called false positives.

"*Though many attempts have been made to measure robustness of face recognition algorithms, there has been no method that has proven adequate.*" (Little et al, 2005:89)

And problems are not only evident with facial recognition technology but also in the performance evaluation of these systems. In their Best Practices in Testing and Reporting Performance of Biometric Devices (Mansfield &.Wayman, 2002) reveal a wide variety of conflicting and contradictory testing protocols. (Mansfield & Wayman, 2002) further suggest that there is a need for a common testing protocol and goes on to provide some general best practice concepts in the development of scientifically sound test protocols for the field of biometrics. What is clear is although some efforts towards the provision of effective performance evaluation of biometrics have been successful a comprehensive performance evaluation framework for image processing algorithms is still absent.

The next section reports on the barriers faced in developing such an effective performance evaluation framework for IP algorithms.

2.6 BARRIERS TO COMPREHENSIVE IP ALGORITHM PERFORMANCE EVALUATION

2.6.1 LACK OF METHODOLOGY

As mentioned in Section 2.5.2, apart from biometrics little emphasis has been put on algorithm performance evaluation standards up to now and where evaluation has taken place, it is been carried out in a somewhat cumbersome and unsystematic fashion, without any standardised approach (Thacker et al, 2005:35). In 1997 (Micheals & Boult, 2001:150) reported no efficient statistically sound approach for evaluating general classification or recognition systems was in existence. Similarly in 1998 the (9th Workshop "Theoretical Foundations of Computer Vision" Evaluation and Validation of Computer Vision Algorithms, 1998) found that although in certain areas of computer vision a surplus of literature was available on the subject, the research community still faced a lack of well grounded and standardised algorithm evaluation methodology.

"*Although the evaluation and validation of algorithms have been discussed for over a decade, the research community still faces a lack of well-defined and standardized methodology.*" (Takeuchi et al, 2003:408)

Over the course of the last 10 years not a lot has changed with (Thacker et al, 2005:35) reporting that apart from the field of biometrics there is still only inconsistent use of performance techniques by researchers. In 1999 (Liu & Dori, 1999:97) pointed out that although research on performance evaluation on specific classes of recognition algorithms had taken place such as (Phillips et al, 1998), a standard methodology that can be applied to all graphics recognition algorithms was currently unavailable. (Liu & Dori, 1999:97) found at the time that even current performance evaluation methodologies for specific classes of recognition algorithms were not up to requirements. Furthermore (Liu & Dori, 1999:98) reported that there was no common methodology for performance evaluation and although ground truth was widely accepted the definitions of comparing ground truth with recognised results from algorithms and the selection of the appropriate performance metrics were still undefined and controversial.

"To further advance the research on graphics recognition, to fully comprehend and reliably compare the performance of graphics recognition algorithms, and to help select, improve, and even design new algorithms to be applied in new systems designed for some specific application, the establishment of objective and comprehensive evaluation protocols and a resulting performance evaluation methodology are strongly required." (Liu & Dori, 1999:97)

In the area of face recognition the lack of a common benchmark database and comparison framework has been identified by (Little et al, 2005; Lu, 2003). (Zhao et al, 2000; Zhao et al, 2002) points out that with the numerous face recognition theories and techniques now available, evaluation and benchmarking of these algorithms are crucial.

"*Computer vision researchers have emphasized the need to find methodologies to characterize the performance of face recognition algorithms, which deal with a huge amount of high dimensional data.*" (Little et al, 2005:89)

Where performance evaluation has taken place to measure face recognition rates, (Little et al, 2005:89) finds that none of these attempts studied the algorithms under varying pose or illumination conditions – the primary sources of difficulties for face recognition (Dalong et al, 2005:793;Kim et al, 2003:29). In 2005 (Little et al, 2005) summarised that although many attempts have been made to measure robustness of face recognition algorithms no methods have proven satisfactory. Similarly (Low & Hjelmas, 2001; Sharma & Reilly, 2003) highlights the lack of standard performance evaluation measures for face detection purposes and (Sharma & Reilly, 2003) proposes a method (including a face image database) for the evaluation and comparison of existing face detection algorithms. Even with regard to the most common customer complaint in the digital imaging market, (Ulichney et al, 2003) highlights the need for automated red eye correction and detection algorithm testing, pointing out that algorithm enhancement depends on known failures.

In brief, as (Clark & Clark, 2002; Heo et al, 2003:558; Liu & Dori, 1999:98; Sharma & Reilly, 2003) report, a common methodology for the performance evaluation of detection and recognition algorithms is needed. However confusion over best practice and standard terminology on how this methodology should be implemented are implicit in the

literature. In 1998 (Chhabra & Phillips, 1998) described a testing procedure for IP algorithms that automatically analyse engineering drawings incorporating a testing protocol and a system for generating ground truth. In (Liu & Dori, 1999:105) recommend the establishment of a performance evaluation protocol with three essential elements:

1. Ground truth and a sound methodology for acquiring it
2. Matching definition where each ground truth graphic object must first be matched with one or more objects from the recognised objects set and so should be in the same format.
3. Representative metrics of interest should be selected to display findings.

In 2003 (Sharma & Reilly, 2003) suggested any evaluation procedure should use standard terminology along with carefully labelled face databases for evaluation purposes. (Sharma & Reilly, 2003) also propose that any performance results should be presented graphically to facilitate fast and effective interpretation.

"*They may help compare, select, improve, and even design new methods to be applied in new systems designed for some specific application.*" (Hua et al, 2004:498)

Obviously as (Yang et al, 2002:54) reports any fair and effective solution needs careful design of protocols, scope, and datasets. As mentioned in Section 2.5.6 the only feasible way to compare algorithms is to run them on the same data. Most performance evaluation techniques involve the creation and running of test scenarios incorporating the IP algorithm on sample data sets to deduce performance scores (Clark & Clark, 2002:15; Courtney & Thacker, 2001). Consequently as mentioned in Section2.5.6 and inline with (Clark & Clark, 2002:4) principles each individual algorithm test scenario requires three pieces of information:

- **Test Data**
- **Ground Truth**
- **Metrics of Interest**

The next section investigates the barriers in implementing a performance evaluation framework based on the three essential components of an image processing algorithm test scenario mentioned above.

2.6.2 Test Data – Lack of Comprehensive Image Database

"*There are many challenges to building a comprehensive face database.*" (Black et al, 2002:11)

One major requirement of an effective algorithm testing methodology is the provision of a large database of test images to allow for comprehensive evaluation. (Ulichney et al, 2003) suggests,

"*The success and quality of many image enhancement algorithms, specifically those that address a particular image defect such as red-eye, depend on the breadth of test cases available to measure performance.*" (Ulichney et al, 2003:1)

(Kong et al, 2005:111) report that any database must contain a large number of test images for adequate assessment and (Moon et al, 2002:7) states that ideal performance evaluation requires all possible images to be tested. In essence because numerous imaging conditions can affect the performance of an IP algorithm (Moon et al, 2002:1) reports all detection algorithms require a large database of images for both training and testing purposes.

"*All current face recognition algorithms fail under the varying conditions in which humans can and must identify other people.*" (Pentland & Choudhury, 2000:55)

Unfortunately (Courtney & Thacker, 2001; Yang et al, 2002:49; Zhao et al, 2000) report although numerous face recognition algorithms exist most of them have not been tested on datasets with a large number of images and (Sharma & Reilly, 2003) points out that the lack of a standard face detection database has caused a lot of algorithm comparison difficulties. (Clark & Clark, 2002:5) reports that the datasets used in performance evaluation are not large or comprehensive enough to adequately test the algorithm, while (Takeuchi et al, 2003:409) finds a large number of papers report excellent performance of their algorithms, based on small data sets.

Consequently, many algorithms developed have not been tested on an image database containing test sets which possess a high degree of variability in terms of scale, location, orientation, pose, facial expression lighting conditions. Problems with two of the most common databases, the FERET (Phillips (b) et al, 2000) and MIT databases used for face recognition evaluation are systematic of problems across all algorithm evaluation frameworks. Although the FERET database has aided face recognition evaluations (Black et al, 2002:4; Kong et al,2005: 128; Little et al, 2005:89) finds that it does not possess a wide enough variety of illumination and pose variations or information on the lighting used to capture the images. Similarly (Black et al 2002:5) finds although the MIT database contains images that were captured with different pose and lighting variations, these variations are not extensive enough. The MIT database (Yang et al, 2002) reports, only consists of frontal and near frontal view images on a cluttered background. Such databases do not provide the challenges that face detection algorithms can encounter in real applications, such as poor image quality, presence of multiple faces and faces with different orientations (upright and rotated).

"Though efficient and robust face detection algorithms [6, 16, 18] have become available, the effectiveness of available face recognition algorithms is still limited to images of mug shots in which faces are mostly in frontal and with reasonably homogenous lighting conditions and small variations in facial expressions [3, 12, 22]." (Zhang et al, 2004a:1)

As mentioned in Section 2.4, numerous imaging conditions can affect the performance of an algorithm, including camera and lighting conditions, so it is important to identify which parameters affect the criterion function significantly and which do not. Poor performance of an IP algorithm is usually going to be dominated by the parameters which occur repeatedly and/or have a strongly harmful effect on image quality. (Moon et al, 2002:2) find that the camera and illumination angles are the most crucial factors. (Moon et al, 2002:1; Ojala et al, 2002:705) report that if these factors are quantified and if very large set of images annotated with these parameters are available, testing on this set of images would give a very informative performance measure for the algorithm.

"The estimation of one or more of these appearance parameters from one or more images of the scene has been an important part of research in computer vision" (Narasimhan, 2002:148)

Another factor in the need for category-based test sets is due to the difference in IP algorithm performance on images taken by different image capturing devices. According to (Phillips et al, 2000a:61) this variation has the potential to affect algorithm performance as severely as changing illumination. Unfortunately, unlike the effects of changing illumination, (Phillips et al, 2000a:61) reports the effects on performance of using multiple camera types has not been quantified. Therefore a means of categorising multiple images based on different cameras types is certainly needed.

As awareness of the above mentioned difficulties have grown, more and more researchers have attempted to address these problems. In 2001 (Gross et al, 2001) reported that large databases with ethnic variations were available; however they lack the variation in lighting, shape, pose and other factors. In 2003 (Sharma & Reilly, 2003) developed an image database containing colour images providing 'real world' challenges to face detection algorithms by *"including faces with a large variety in size, shape, orientation, expression and images that have varying lighting conditions, resolution and backgrounds."* (Sharma & Reilly, 2003). Unfortunately, however as (Black et al, 2002) reports regards regarding face databases, most of the currently available image databases in use today are not adequate to achieve comprehensive performance evaluation of the IP algorithms.

2.6.3 Lack of a Systematic Means Of Acquiring Accurate Ground Truth

"Given a large number of ground truth data sets from different environments, statistical evaluations are possible as well as the robust assessment of performance of algorithms." (Takeuchi et al, 2003:409)

An important feature of any evaluation procedure is the necessity to determine the appropriate characteristics of the input data. According to (Micheals & Boult, 2001:152) one of the fundamental difficulties faced in algorithm performance evaluation is the difficulty of acquiring sufficient data. In 1999 (Liu & Dov, 1999) reported that the importance of accurate ground truth had become widely accepted essentially because as (Müller et al, 2004) reported in 2004, an IP algorithm is only as good as the database and

ground truth it is tested upon. Ground truth is the corresponding expected output from the algorithm determined by a human which is compared with the actual output to measure performance. In a lot of cases this takes the form of a content-based feature.

"*A feature is defined as a descriptive parameter that is extracted from an image or video stream*" (Bovik, 2000:689)

(Bovik, 2000:689) define a content-based feature as a feature that is derived for the purpose of describing the actual content in an image (Bovik, 2000:689). Clearly because each ground truth content-based feature must be matched with one or more features recognised by the IP algorithm, both should be in the same format (Liu&Dori, 1999)

"*Visual data is usually more complex than the data typically analyzed in statistics, and so often a straight forward application of robust statistical techniques does not work.*" (Meer et al, 2000:2)

Since most IP algorithm performance relies on ground truth accuracy (Takeuchi et al, 2003:413), its gathering or generation is usually a tedious and onerous task (Chhabra & Phillips, 1998; Micheals & Boult, 2000:1; Van House et al, 2004a). (Liu & Dov, 1999) further point out that manual ground truth acquisition is somewhat subjective and can vary from one human to another. Evidently as (Chhabra & Phillips, 1998) suggested in 1998 and still up to now there is a lack of a systematic way of generating accurate ground truth.

2.6.4 Lack of Standard Terminology And Poor Use Of Metrics Of Interest

According to (Courtney & Thacker, 2001:4) in order to carry out a test, it is necessary to define a metric that can be used to quantify performance. (Micheals & Boult, 2000:3) reports algorithm evaluation seeks the algorithm that yields the most desirable behaviour, as dictated by a set of metrics. And if each algorithm is presented the exact same inputs, the only system variation is the algorithm. A metric is a criterion function which quantitatively measures the difference between the ideal output arising from the perfect ideal input and the calculated output arising from the corresponding randomly perturbed input. With regards face detection which can be viewed as a two-class (face versus non-

face) classification problem (Hsu et al, 2001; Kong et al, 2005:109), most papers compare performance of face detection algorithms in terms of detection rates and false alarm rates. In 1999 (Phillips & Chhabra, 1999) defined a measurement for the performance (accuracy) of a detection algorithm by counting the number of matches between the entities detected by the algorithm and the entities in the ground-truth, and the number of misses and false-alarms. (Phillips & Chhabra, 1999) defined several system performance metrics including:

Detection Rate is, roughly, the percentage of ground-truth entities that are detected by the recognition system.

Missed detection rate is the percentage of ground-truth entities not detected by the recognition system.

False-alarm rate is the percentage of detected entities produced by the system that do not match with any entity in the ground-truth.

Recognition accuracy indicates, roughly, the percentage of detected entities with the result file that have their match in the ground truth entities. Thus, one can consider recognition accuracy as a measurement of the overall accuracy of a recognition system.

Unfortunately the above metrics have not become standard and apart from the FERET evaluation procedures becoming de facto standards in the face recognition field the lack of a standard terminology for performance metrics is evident in the literature. (Sharma & Reilly, 2003; Yang et al 2002:35) reported on the lack of a standard terminology to describe test results meaning different researchers use different definitions for detection and false alarm rates. (Clark & Clark, 2002) in particular report on the confusion in the literature over the terms "false negative" and "false positive". Consequently this in turn has led to difficulties in comparing algorithms.

As can be seen from the above section testing image processing algorithms effectively is a slow and laborious process. Algorithms should be executed on a set of image-test-sets

with the appropriate variation in operating conditions. And output should be compared with accurate ground truth using proper metrics of interest to deduce relevant performance scores. Accordingly and inline with (Clark & Clark, 2002:12) recommendations, the best way to perform testing, especially when the results are to be used for comparison, is to use a specially-designed software package. (Clark & Clark, 2002:12) defined it as a "Test Harness" and essentially it means automating the testing process of image processing algorithms. The next section outlines the key components of such a "Test Harness", an effective image processing algorithm performance methodology.

2.7 RECOMMENDATIONS FOR EFFECTIVE IP PERFORMANCE EVALUATION

2.7.1 AN EFFECTIVE IP PERFORMANCE EVALUATION METHODOLOGY.

"*Scenario evaluations in machine vision often result in the problem of establishing how well an algorithm can identify a particular situation in the image.*" (Thacker et al, 2003:29)

As image processing algorithms are designed to solve a specific task (e.g. detect face in image), evaluation is also task dependent. The variety of image processing tasks therefore leads to a variety of different requirements for not only each class of algorithm but for each individual algorithm. For instance one face detection algorithm may require a different set of input parameters to that of another. Therefore no single set of performance metrics or constraints can be applied to all algorithms (Föerstner, 1996:3).

"*Measures easily derivable for one algorithm may not be derivable for another one.*" (Föerstner, 1996:12)

Therefore in accordance with (Clark & Clark, 2002) principles the most effective way of evaluating IP algorithm performance is by means of running the IP algorithm on a large set of input data whose correct outputs are known and comparing the resulting output from the algorithm with known correct results. This is in accordance with the FERET evaluation methodology (Phillips et al, 2000b) where there is a direct connection among

the problem being evaluated, the test-image-sets, and the actual testing protocol. According to (Phillips et al, 2005:948) this allows researchers to assess the best approaches and fine-tune their algorithms. Consequently test execution, the actual running of an algorithm on the image-test-set should be the central point of any algorithm performance evaluation methodology. In essence it means automatically running the image processing technique on a designated set of marked images. Therefore the recommended algorithm performance evaluation methodology will have three major requirements.

2.7.2 Test Data – A Comprehensive Database Supplying Image-Test-Sets

The first key requirement is the test data, the input to the algorithm under test. From a conceptual viewpoint, digital image processing revolves around digital images. Evaluating an IP algorithm's performance in this way involves the execution of the algorithm on a large set of images - the image-test-set. Therefore the image-test-set, which the algorithm is evaluated upon, is crucial. As previously discussed in Section 2.6.2, for the most part the image-test-sets currently used in performance evaluation are inadequate. (Clark & Clark, 2002:5; Courtney & Thacker, 2001; Müller et al, 2004; Sharma & Reilly, 2003, Ulichney et al, 2003; Yang et al, 2002:49; Zhao et al, 2000) all highlight major shortcomings with standard face detection, face recognition and red eye removal image databases. Therefore it is imperative that any new image database should contain a large number of standard and representative test images for adequate assessment (Kong et al, 2005:111; Moon et al, 2002:7; Yang et al, 2002:52).

Poor performance of an algorithm is usually going to be dominated by the parameters which occur repeatedly and/or have a strongly harmful effect on image quality. For instance (Dalong et al, 2005:793; Kim et al, 2003:29) identify the primary sources of difficulties for face recognition as varying pose or illumination conditions. By categorising image-test-sets (in which all but one parameter is held constant) it becomes feasible to evaluate an IP algorithms ability to tolerate changes in each type of

environmental variable (Black et al, 2002). Another factor in the need for category based test sets is due to the difference in algorithm performance on images taken by different image capturing devices. According to (Phillips et al, 2000a:61) this variation has the potential to affect algorithm performance as severely as changing illumination. Consequently any effective performance evaluation methodology should allow for efficient categorisation of test images permitting the IP algorithm to be tested on the most relevant image-test-set.

2.7.3 Need for Accurate Ground Truth And An Effective Methodology For Acquiring It

The second key requirement of an effective algorithm performance evaluation is the acquirement of accurate ground truth data (Black et al, 2002; Liu & Dov, 1999; Müller et al, 2004). Ground truth is compared with the actual output from the algorithm test run results to evaluate algorithm performance. Therefore any performance evaluation methodology should provide a tool to generate a large database of ground truth for evaluating segmentation, classification and recognition algorithms. And as (Liu & Dori, 1999) suggests, because each ground truth object must be matched against one or more objects from the recognised objects set, both should be in the same format. Then if the marking system generates accurate markings it becomes a simple matter of comparing ground truth markings with actual output from the algorithm test run to determine the percentage of false positive and false negative results of each algorithm and the correctness of the detected positions and shapes of each object.

2.7.4 Use Of Appropriate Metrics Of Interest

The third key requirement is the definition of and use of appropriate metrics of interest. Performance evaluation is not just finding out whether algorithms perform as expected; according to (Courtney & Thacker, 2001), it involves the use of objective, usually statistical measures for comparing the performance of vision algorithms (Courtney & Thacker, 2001; Micheals & Boult, 2000). Accordingly metrics are usually employed to

express the difference between the expected output (ground truth) and the actual output (returned results) from the algorithm test (Hua et al, 2004:498, Liu & Dori, 1999). Different IP algorithms require different performance measures so the effectiveness of any algorithm representation depends critically upon selecting the appropriate parameters to describe the performance metrics.

Unfortunately, apart from the FERET evaluation procedures which have become de facto standards in the face recognition field (Phillips et al, 2000b), the lack of a standard terminology in selecting and defining performance metrics is evident in the literature. Consequently in line with principles set out by (Sharma & Reilly, 2003) any evaluation procedure should use standard terminology for evaluation purposes. Additionally since different IP algorithm tasks require different performance measures, any comprehensive performance evaluation methodology should define and use the appropriate metrics for the particular class of algorithm being tested. Ultimately, this core component of the performance evaluation methodology will allow for developers to pinpoint the problems within the algorithms they are working on quickly and more effectively.

2.8 CHAPTER SUMMARY

This chapter started by introducing the everyday use of digital image processing technology in consumer appliances and its increasing importance to other areas of industry including security. After a selective review of popular image processing algorithms in use today, the main problems in image processing algorithm development and testing are presented. A comprehensive survey on the methods and techniques being used in image processing algorithm performance assessment was then undertaken and the need for an efficient image processing algorithm testing framework was identified. Finally, the main components of a comprehensive image processing algorithm testing methodology were outlined, that will aid in assessing and improving the quality of existing and newly developed algorithms.

CHAPTER 3: METHODOLDOGY AND REQUIREMENTS ANALYSIS

Methodology and Requirements Analysis

3.1 INTRODUCTION

As digital image processing techniques have become increasingly used in a broad range of applications (Gonzalez & Woods, 2002:6, Jaynes et al, 2005:1), the critically need to evaluate algorithm performance has become recognised by developers as an area of vital importance (Courtney & Thacker, 2001; Hua et al, 2004:498; Meer et al, 2000:2 Micheals & Boult, 2001:150). After conducting a survey of the literature on the methods and techniques being used, it can be seen that current algorithm testing and evaluation practices do not live up to expectation. Currently no standard methodology exists for the performance evaluation of detection and recognition algorithms (Clark & Clark, 2002; Heo et al, 2003:558; Liu & Dori, 1999:98; Sharma & Reilly, 2003) and although techniques have been developed for assessing individual classes of algorithms (Liu & Dori, 1999:97; Phillips et al, 2000b), to date a generalised methodology for image processing algorithm performance assessment has not been developed.

"Although the evaluation and validation of algorithms have been discussed for over a decade, the research community still faces a lack of well-defined and standardized methodology." (Takeuchi et al, 2003:408)

Only in the highly financed field of biometrics has any real progress been made in testing algorithms effectively (Thacker et al, 2005:35). As a result, algorithms are often implemented based on programmers' intuition and experience rather than any standard performance evaluation. It is only extensive testing that proves an IP algorithm's accuracy and it is this very process that can speed up, or slow down algorithm development. A new approach that incorporates a comprehensive testing methodology

and framework to adequately measure algorithm performance would greatly improve the quality and efficiency of the algorithm testing process.

The next section outlines the key aims and objectives of a proposed testing methodology to assist developers in measuring image processing algorithm performance effectively.

3.2 OBJECTIVES OF PROPOSED IP ALGORITHM TESTING METHODOLOGY

By enforcing adherence to a consistent test approach the new methodology aims to:

1. Enable the analysis and performance assessment for a wide range of the image processing algorithms in existence today (Jaynes et al, 2005:1; Kim et al, 2004; Little et al, 2005:89; Pentland & Choudhury, 2000:51; Yang et al, 2004:2533).

2. Provide easy access to a comprehensive set of relevant test images that will allow algorithms be tested on the most pertinent image-test-sets for a more complete evaluation (Jaynes et al, 2005:2).

3. Demonstrate how the analysis of image processing algorithms can be considered a relatively simple function by decoupling the algorithm from the operating conditions, gathering the appropriate data, and then categorising the problem according to a set of specified criteria (Sharma & Reilly, 2003).

4. Deduce algorithm performance with an high degree of accuracy by providing effective tools for accurate and efficient ground truth acquirement.

5. Supply the required algorithm performance data which can then be used to optimise algorithm performance (Pankanti et al, 2000:49) by providing a means of specifying relevant metrics of interest for performance analysis.

6. To facilitate fast and effective algorithm performance analysis, performance data presented graphically should have three fundamental requirements (Sharma & Reilly, 2003):
   - Performance data should be easy for both novice and expert user to understand.
   - Allow for adequate assessment of algorithm past performance by enabling the history of algorithm improvements iterations to be displayed.
   - Allow for investigation of algorithm performance on individual images as well as quantitative analysis of image-test-set performance.

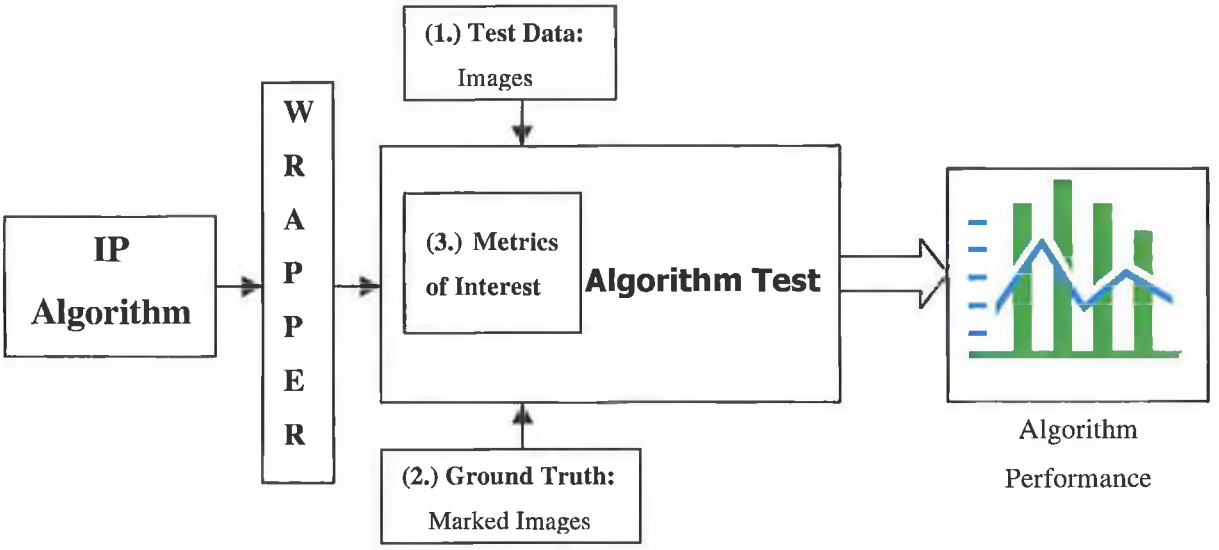
7. Speed up the testing process by providing an integrated testing environment resulting in faster and more efficient testing practice.

Beyond the above, the methodology should play a critical role in guiding and focusing image processing algorithm development research. Based on the objectives of the proposed IP algorithm testing methodology outlined above, the next section gives an overview of the proposed image processing algorithm test framework and its key components.

3.3 THE PROPOSED IP ALGORITHM TEST FRAMEWORK OVERVIEW

Figure 4 provides an overview of the proposed IP algorithm test framework and the key inputs of the IP algorithm testing methodology.

Figure 4: IP Algorithm Test Framework Overview



The first key input into the IP algorithm testing methodology is the IP algorithm to be tested. Since the proposed test framework will provide for the performance assessment of a wide range of the IP algorithms, an interface known as a wrapper based on the individual IP algorithm being tested is required. This wrapper will allow the testing of various IP algorithms within the test framework.

After the IP algorithm has been selected for testing and inline with (Clark & Clark, 2002:4) recommendations, each individual IP algorithm test scenario requires three pieces of information. As illustrated in Figure 4, the first key requirement is the **Test Data**, the input to the algorithm under test. In most cases this takes the form of a relevant set of test images. The concept behind this is that high-quality performance on the set of images should correspond to high-quality performance on similar or corresponding images the algorithm will encounter in its real world application (Kong et al, 2005:111; Thacker et al, 2003:28; Thacker et al, 2005:28). The second key requirement is the acquirement of accurate and relevant **Ground Truth** data for the test (Micheals & Boult,

2001:152; Müller et al, 2004). Ground truth is the corresponding expected output from the IP algorithm determined by a human and must be compared with the actual results from the test run - the algorithm output to evaluate algorithm performance. It usually takes the form of the co-ordinates of manually marked content features within the images based on the type of IP algorithm being evaluated. Finally the third key requirement is the specification of relevant **Metrics of Interest** to express the difference between the expected output (ground truth) and the actual output (returned results) from the IP algorithm test (Hua et al, 2004:498, Liu & Dori, 1999). By allowing the specification of metrics of interest, the desired IP algorithm performance characteristics can be isolated and evaluated by the algorithm developer.

The first step in building the IP algorithm test framework outlined above is the specification and analysis of the user's requirements in order to implement an approach that fully satisfies the user goals. The key functional requirements of the proposed software solution are outlined in next section

3.4 FUNCTIONAL REQUIREMENTS SPECIFICATION

"Any system designed for people to use should be easy to learn (and remember), useful, that is, contain functions people really need in their work, and be easy and pleasant to use" (Gould & Lewis, 1985:300)

Functional requirements identify what the system should do (Preece et al, 1994). Based on the key elements of the proposed IP algorithm test framework outlined above, the next section lists the functional requirements of the proposed testing methodology in detail.

3.4.1 Integrating Different IP Algorithm into the IP Algorithm Test Framework

One of the most important requirements of the algorithm test framework is extensibility so as to allow for easy integration of various algorithms based on their specific requirements. There are a wide variety of image processing algorithms in existence today (Jaynes et al, 2005:1; Kim et al, 2004; Little et al, 2005:89; Pentland & Choudhury,

2000:51; Yang et al, 2004:2533) each comprising of its own unique characteristics and processing mechanisms (Föerstner, 1996).

"*The variety of tasks leads to a variety of requirements*" (Föerstner, 1996:3)

With the wide array of imaging libraries available, essentially "Plug-and-play" functionality should be provided by the test framework whereby a wrapper is developed to allow integration of an IP algorithm based on its own unique characteristics. In essence the wrapper should allow performance evaluation of specific algorithms designed to do specific tasks - a scenario evaluation (Blackburn, 2001; Courtney & Thacker, 2001:3; Phillips et al, 2000a).

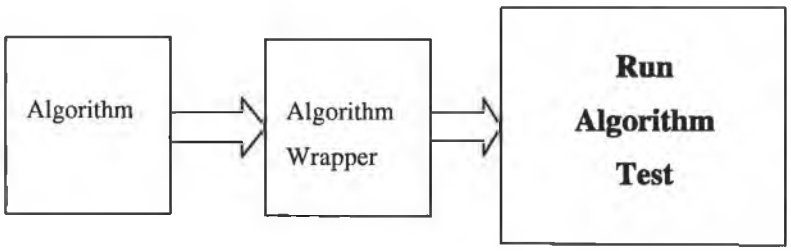


Figure 5: Algorithm Wrapper Overview

3.4.2 IP Algorithm Test Execution: A Test Scenario

Inline with the principles set out in the FERET evaluation methodology (Phillips et al, 2000b), the proposed methodology should exploit the direct connection between the algorithm being evaluated, the set of test images and the actual testing protocol. For instance it is of little use testing a red eye removal solution on an image-test-set of landscape images. The proposed framework should therefore allow individual test scenarios to be specified based on algorithm and developers requirements. A test scenario will allow the definition of what parameters the algorithm takes as input, what output the algorithm returns and how this is compared with ground truth to deduce performance scores (Föerstner, 1996:12).

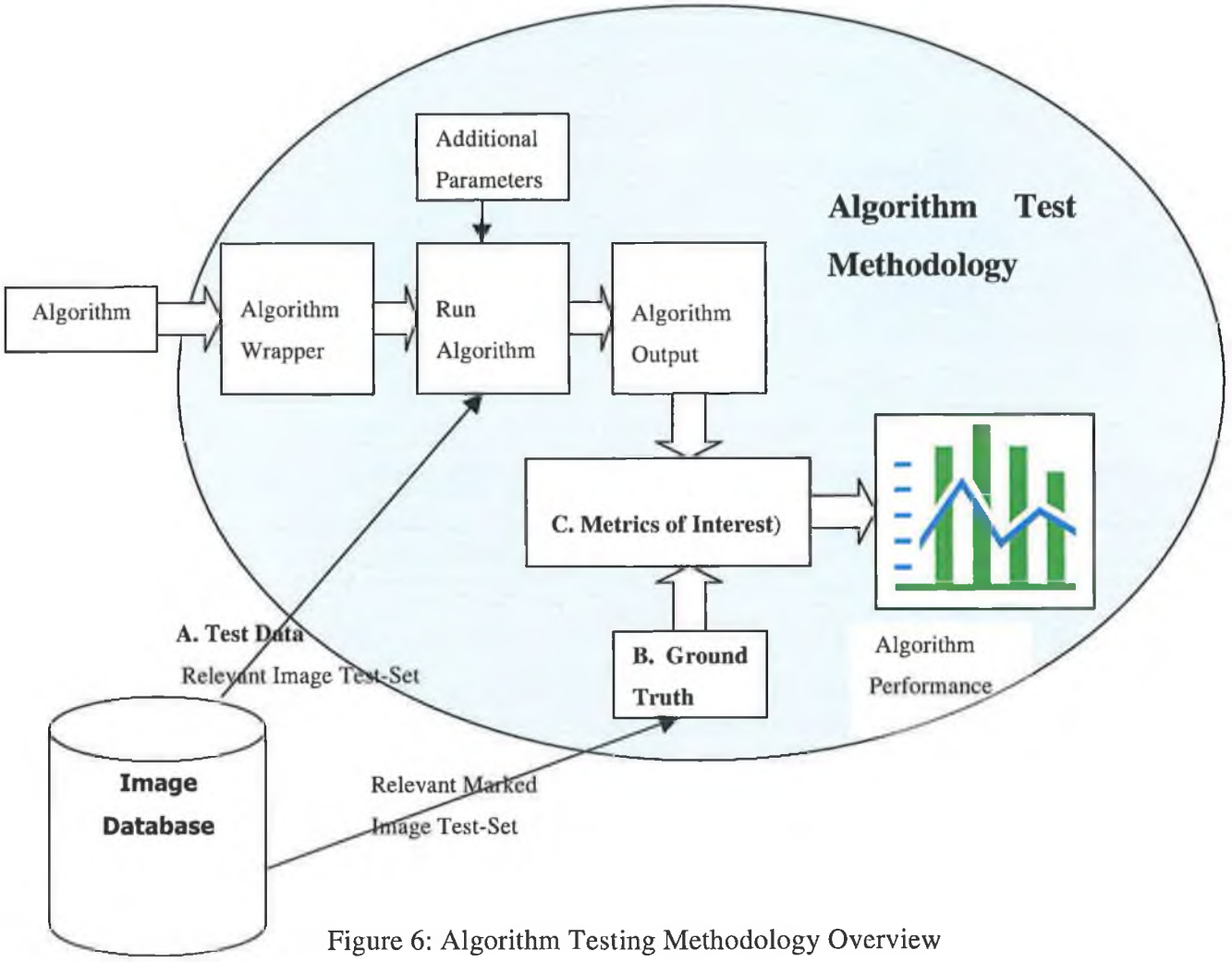


Figure 6: Algorithm Testing Methodology Overview

In essence the test scenario will incorporate the following components (Clark & Clark, 2002:4; Liu & Dori, 1999:105):

A. The **Test Data** is the actual input to the algorithm under test (Image-test-set).

B. The **Ground Truth** is the corresponding expected output from the algorithm determined by a human.

C. Whether the output corresponds to a success or failure (Liu & Dori, 1999:98). A matching method and representative **Metrics of Interest** to evaluate performance.

3.4.2.1 Test Data: Image-Test-Set

"*One must explore what characteristics of the inputs affect the algorithms performance and how much*" (Clark & Clark, 2002:3).

A vital part of the testing process is the set of images, which the IP algorithms are tested on. These are known as image-test-sets. The concept behind this is that high-quality performance on a representative set of images should correspond to high-quality performance on similar or corresponding images the algorithm will encounter in its real world application (Kong et al, 2005:111; Thacker et al, 2003:28; Thacker et al, 2005:28). Therefore the provision of a database containing image-test-sets which the algorithm is evaluated upon is crucial. Up to now such a database has not been available to algorithm testers. Many of the successful claims in the literature have used private image-test-sets that are not comprehensive or varied enough leading to biased and erroneous results (Takeuchi et al, 2003:409). And over the last decade the need for large databases of image-test-sets to allow for comprehensive evaluation of face detection, face recognition and red eye removal solutions in particular has been identified in the literature (Black et al 2002:4; Clark & Clark, 2002:5; Courtney & Thacker, 2001; Gross et al, 2001; Hsu et al, 2001; Little et al, 2005:89; Müller et al, 2004; Sharma & Reilly, 2003; Ulichney et al, 2003:1; Yang et al, 2002:49; Zhao et al, 2000). Therefore in order to validate and test algorithms comprehensively, a key requirement of the proposed test framework is the image database which should incorporate some vital features:

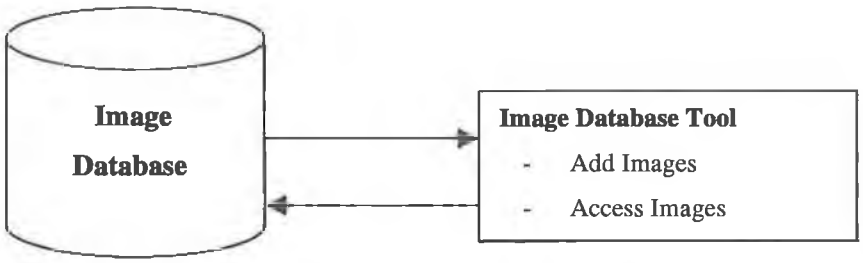


Figure 7: Image Database Tool Overview

3.4.2.1.1 As outlined in the previous chapter, numerous imaging conditions may affect the performance of a IP algorithm, so to reveal the strengths and weakness of an algorithm it must be run on pertinent image-test-sets (Jaynes et al, 2005:2; Kong et al, 2005:111; Moon et al, 2002:7; Yang et al, 2002:52). The image database accordingly should contain a large number of standard and representative test images which possess a high degree of variability in terms of scale, location, orientation, pose, facial expression, lighting conditions, etc.

3.4.2.1.2 To speed up the algorithm testing process the database ought to provide a means for the categorisation of multiple images based on image content and properties. This allows for the natural selection of the most relevant image-test-sets for algorithm test runs. To allow for fast and efficient access the image database should categorise images by different parameters based on image content (Bovik, 2000:689) and properties. Poor performance of an algorithm is usually going to be dominated by the parameters which occur repeatedly and/or have a strongly harmful effect on image quality. Most currently available test image databases are not adequate to evaluate tolerance of variations in environmental parameters such as the PIE (Pose, Illumination, Expression variant) problem (Dalong et al, 2004:787). The proposed test framework should allow for factors such as these to be quantified in image-test-sets. Testing on such image-test-sets will give the required informative performance measure for the algorithm (Moon et al, 2002:1; Ojala et al, 2002:705). Another factor in the need for category based image-test-sets is due to the difference in algorithm performance on images taken by different image capturing devices. This variation has the potential to affect algorithm performance as severely as changing illumination (Phillips et al, 2000a:61).

3.4.2.1.3 The image database should also reflect the new and unique challenges facing algorithm testing. As new algorithm techniques emerge that utilise different image

properties and content (Nishino et al, 2005(a); Pankanti et al, 2000:49) new image-test-sets must also be produced to rigorously evaluate these algorithms.

3.4.2.2 Ground Truth: Marked Image-Test-Set

A second key requirement of an effective algorithm performance evaluation is the acquirement of accurate and relevant ground truth data for the test (Micheals & Boult, 2001:152; Müller et al, 2004).

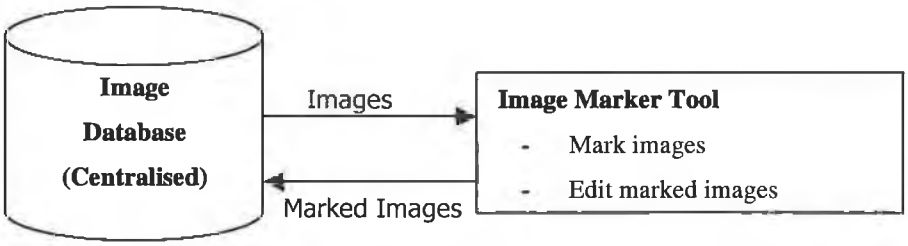


Figure 8: Image Marker Tool Overview

3.4.2.2.1 A large database of ground truth for evaluating segmentation, classification and recognition algorithms corresponding to the selected image-test-set is a key input to the test methodology. Ground truth in this case being relevant features of interest marked or annotated on the selected image-test-set in the same format as that which the algorithm under test will return as output (Liu & Dori, 1999). It then becomes a simple matter of comparing the corresponding ground truth with actual output from the algorithm under test (Phillips et al, 1998) to determine algorithm performance. Therefore the proposed IP algorithm test framework should provide an efficient means of generating accurate ground truth.

3.4.2.2.2 In addition a key component of the proposed ground truth (markings) lies in its complementary information (Sarvas et al, 2004:36). When images are marked, the markings - or ground truth data – should be stored for future use. Ground truth metadata, literally information about information, may not only be used for the purpose of algorithm evaluation, it can also be used as a means of categorising images within the

image-test-set database (Section 2.6.2) (Girgensohn et al, 2003; Girgensohn et al, 2004b; Zhang et al, 2003; Zhang (a) et al, 2004).

Note: Although relevant image-test-sets and accurate ground truth are essential requirements of the algorithm testing methodology, the development of software tools for the gathering, storage and marking of images is not part of this thesis. This work is carried out by other researchers, working on the "Tools & Algorithms to Assist in Automatically Recognising and Deducing Information about People in Consumer Digital Images" Enterprise Ireland (EI) funded project.

3.4.2.3 Metrics of Interest

Test scenario execution, the actual running of an algorithm on the image-test-set will be the central point of the algorithm testing methodology. In conjunction with a large database of image-test-sets, comprehensive algorithm testing will allow the execution of the algorithm on the most pertinent image-test-sets. In the case of testing segmentation, classification and recognition algorithms, with ground truth provided in the same format as that which the algorithm under test will return as output it becomes a matter of comparing the two using a matching method specifically tailored to both ground truth and algorithm output format. Accordingly the IP algorithm test framework should allow users to specify metrics of interest to express the difference between the expected output (ground truth) and the actual output (returned results) from the algorithm test (Hua et al, 2004:498, Liu & Dori, 1999). This process will involve defining metrics of interest relevant to the IP algorithm being tested that can be used to quantify performance (Courtney & Thacker, 2001:4; Micheals & Boult, 2000:3).

"*The application of any technology should be based on the careful consideration of sound scientific test results*" (Bone & Backburn 2002:7)

Unfortunately apart from the FERET evaluation procedures becoming de facto standards in the face recognition field (Phillips et al, 2000b) the lack of a standard terminology in selecting and defining appropriate performance metrics is very evident in the literature

(Courtney & Thacker, 2001:9; Liu & Dori, 1999:98; Sharma & Reilly, 2003; Yang et al 2002:35). Therefore the proposed IP algorithm test framework should allow users to specify metrics appropriate to the particular class of algorithm being tested.

3.4.3 Display and Analysis of Test Results

A core part of the proposed testing methodology will be the test-results – the data generated as a result of running the test scenario. As various versions of an IP algorithm are tested on the same set of original images, the methodology should allow test result comparison to clearly show which version is the most efficient. The same image-test-set been used to test each new version. The results of testing different versions of the same IP algorithm should to be saved to allow for statistical analysis of algorithm past performance. With the availability of previous IP algorithm test performance data, performance results can then clearly display which version of the IP algorithm is the most efficient, and whether the new version performs better than a previous version.

Relevant pieces of information from the test-results should be displayed graphically to facilitate fast and effective interpretation (Sharma & Reilly, 2003). For instance algorithm test performance data should be able to be viewed quantitatively to see the overall result for an image-test-set. Additionally algorithm performance on each individual image could be displayed separately to diagnose what discrepancy is causing a skew in test results. Ultimately, this core component of the IP algorithm test framework will allow for developers to isolate and identify performance bottlenecks quickly and build better algorithms (Pankanti et al, 2000:49).

The proposed IP algorithm test framework has a number of advantages over previous solutions and the next section presents a review of the key problems the proposed IP algorithm test framework intends to solve.

3.5 REVIEW OF KEY PROBLEMS SOLVED BY IP ALGORITHM TEST FRAMEWORK

1. The core problem addressed by the framework is how to integrate different image processing algorithms for testing to support a semi-automatic testing processing. The test framework will provide an extensible interface so as to allow the analysis and performance assessment of a wide range of image processing algorithms.

2. The integrated database will provide easy access to a comprehensive set of test images allowing IP algorithms to be tested on the most applicable image-test-sets resulting in a more complete algorithm evaluation (Jaynes et al, 2005:2; Kong et al, 2005:111; Moon et al, 2002:7; Yang et al, 2002:52).

3. The framework will facilitate the decoupling of the algorithm from its operating conditions resulting in a more comprehensive IP algorithm assessment (Moon et al, 2002:1; Ojala et al, 2002:705; Phillips et al, 2000b).

4. The framework will allow for ground truth to be acquired quickly and accurately (Micheals & Boult, 2001:152; Müller et al, 2004).

5. The generic nature of the framework allows for metrics of interest to be specified based on individual algorithm and developer requirements resulting in quicker algorithm performance evaluation.

6. By providing customisable performance results display functionality the framework will allow for performance data to be displayed graphically in a suitable format facilitating fast and effective interpretation by both novice and expert users (Sharma & Reilly, 2003).

7. The integrated nature of the test framework incorporating an image database tool, a ground truth generation tool and a testing tool in the one environment means that algorithm performance is carried out quickly and easily, thus greatly reducing testing time (Clark & Clark, 2002:12)

3.6 CHAPTER SUMMARY

This chapter has outlined the aims and objectives of the proposed testing methodology, a solution to current testing and evaluation difficulties. The requirements of a software solution were first outlined in detail and the set of key functional requirements were then presented. Finally a review of the key problems the proposed testing framework intends to solve was presented. Ultimately, the test framework proposed aims to shorten the algorithm development life cycle by helping to identify algorithm performance problems quickly and more efficiently.

CHAPTER 4: DESIGN

Design

4.1 INTRODUCTION

Having outlined the requirements and defined the functionality of the proposed IP algorithm test framework in the previous chapter this section draws up a blueprint for system implementation called the design. Firstly, the overall algorithm test framework is described and the classification of the two distinct categories of testing tool user is explained. Having formulated the functional requirements of the IP algorithm test framework in the previous chapter, the key non-functional requirements based on the two categorises of user for the testing tool are then identified. After the selection of system development life cycle and technologies are explained the following sections then outline the main design aspects of the testing tool. Firstly section 4.5.1 details the design of the testing tool algorithm integration architecture. Section 4.5.2 then describes the design of the components that make up a test scenario and the underlying test execution architecture and finally Section 4.5.3 details the GUI design process.

4.2 IMAGE PROCESSING ALGORITHM TEST FRAMEWORK

Based on the initial functional requirements outlined in the previous chapter, Figure 9 illustrates a use case diagram of how each specific category of user will interact with the proposed software solution to accomplish a specific task.

"*A use case is a description of a functionality (a specific usage of the system) that the system provides.*" (Eriksson & Penker, 1997:17)

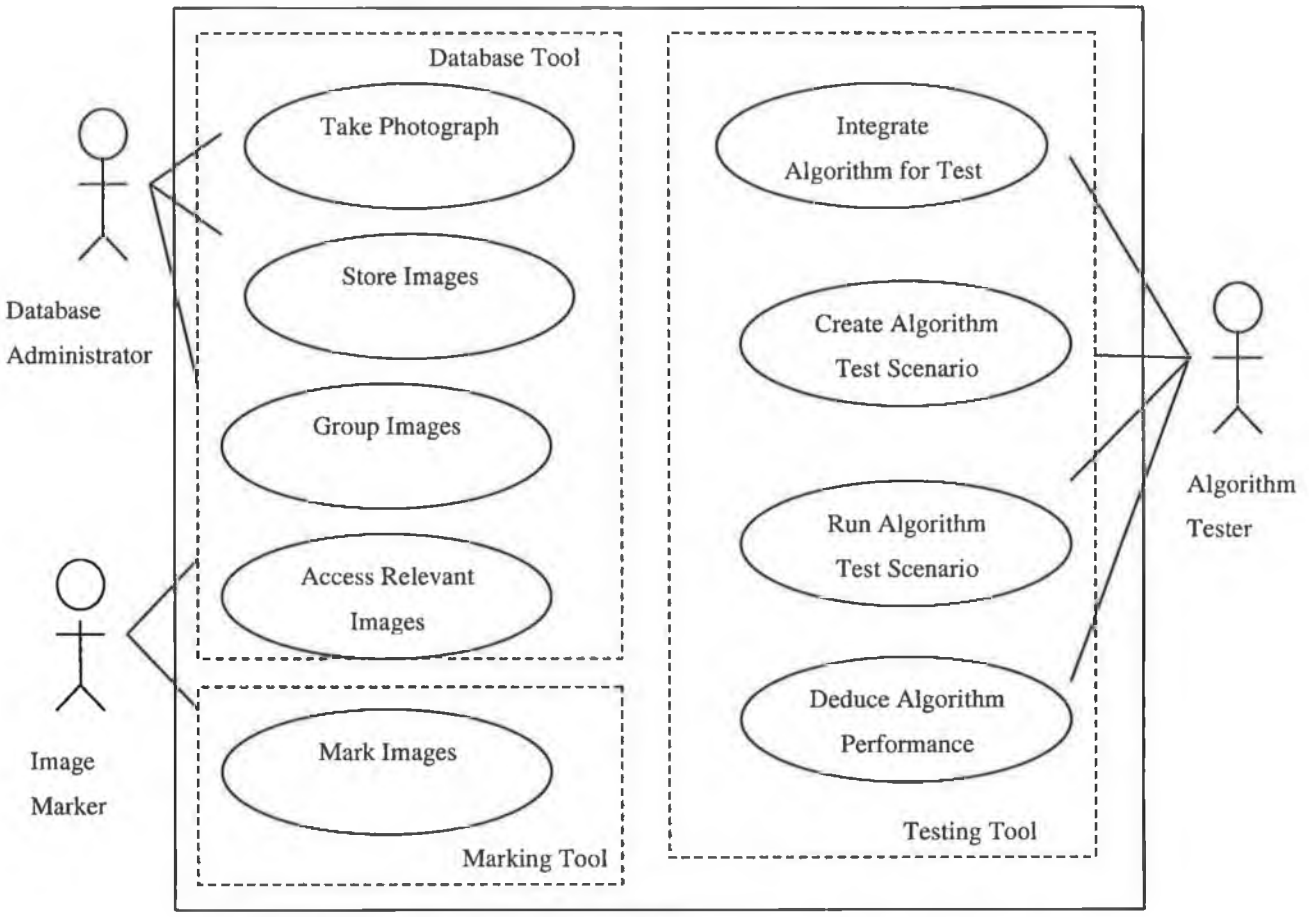


Figure 9: Use Case Diagram for IP Algorithm Test Framework

As illustrated in Figure 9, the three user categories identified are the database administrator responsible for managing image-test-sets, the image marker responsible for generating accurate ground truth data and the algorithm tester who is responsible for managing and executing algorithm tests to deduce algorithm performance. On the evidence of the above use case diagram it becomes possible to decompose the proposed test framework into three separate components outlined in Figure 10.

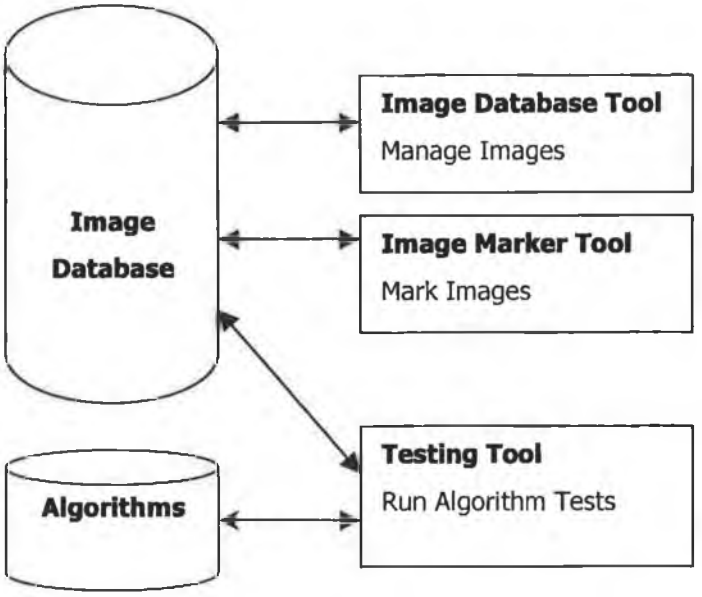


Figure 10: Test Framework Architecture

- The **Image Database Tool** will provide access to a comprehensive set of relevant test images that will allow algorithms be tested on the most applicable image-test-sets for a complete evaluation.

- The **Image Marker Tool** will allow for the accurate and rapid marking or annotation of features of interest within images to be used as ground truth within the testing process.

- The **Testing Tool** allows various algorithms to be plugged into testing tool for performance evaluation. After appropriate metrics of interest have been specified the algorithms can then be run on the most pertinent image-test-sets to deduce algorithm performance.

Given that the thesis's main focus is on the testing tool, the next section further examines the categories of users who will employ the testing tool.

4.2.1 Defining User Categories for the Testing Tool

Figure 11 is a use case diagram that illustrates the different aspects of the testing tool's Functionality and how it is going to be used.

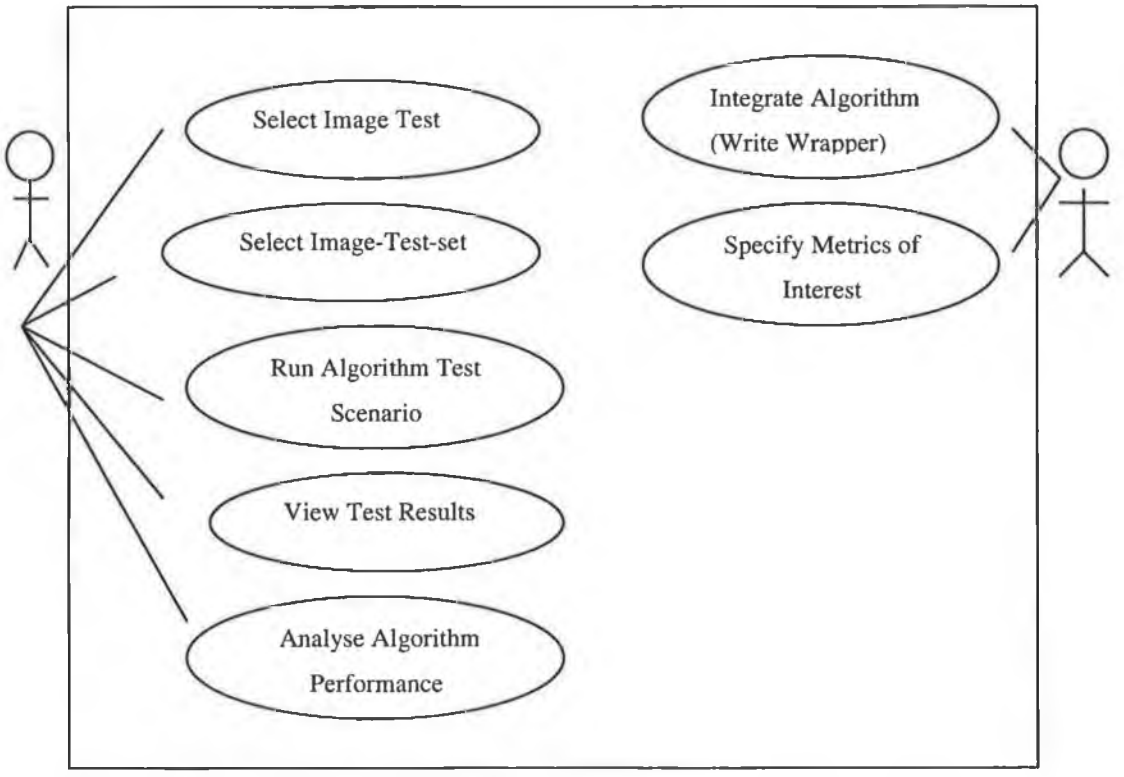


Figure 11: Use Case Diagram for Algorithm Tester and Algorithm Integrator

It is important to note that Figure 11 identifies two distinct classes of user:

- **Algorithm Integrator** who carries out the Algorithm Integration Process.
- **Algorithm Tester** who carries out Algorithm Test Execution.

The algorithm tester, who in most cases is the actual algorithm developer, carries out algorithm test execution to find out whether a new version of an algorithm is better than a previous version. In most cases the algorithm tester will be an expert in the area of algorithm development but may not have the necessary technical knowledge in how to integrate an IP algorithm into the testing tool for performance evaluation. As well become evident, integrating various algorithms for testing purposes and defining relevant

metrics of interest to evaluate algorithm performance is not a trivial task because every algorithm comprises its own unique characteristics and processing mechanisms (Föerstner, 1996).

"*Coding in these languages however can be time consuming because the programmer must iteratively debug compile-time and run-time errors. This approach also requires extensive knowledge of the programming language and the operating system of the computer platform on which the program is to be compiled and run*" (Bovik, 2000:449)

Table 1 for example displays the different characteristics of various IP algorithms.

Image Processing Algorithm Characteristics	**Example**
Designed to perform different tasks	Red Eye Detection Dust Removal
Developed in different programming languages	C++ Java Python
Requiring different inputs parameters	1 Image 3 Images
Producing different outputs parameters or performance data	(True/False) Integer Value

Table 1: Algorithm Characteristics

Therefore it was decided to differentiate the algorithm integration and actual test execution tasks. It makes more sense to allow an expert quickly integrate algorithms into the testing tool without having to retrain individual algorithm developers in the process. The next section identifies the key non-functional requirements of the testing tool.

4.3 NON-FUNCTIONAL REQUIREMENTS OF THE TESTING TOOL

Based on the functional requirements of the IP algorithm test framework, the Quality Requirements Tree illustrated in Figure 12 identifies the key non-functional quality requirements of the testing tool.

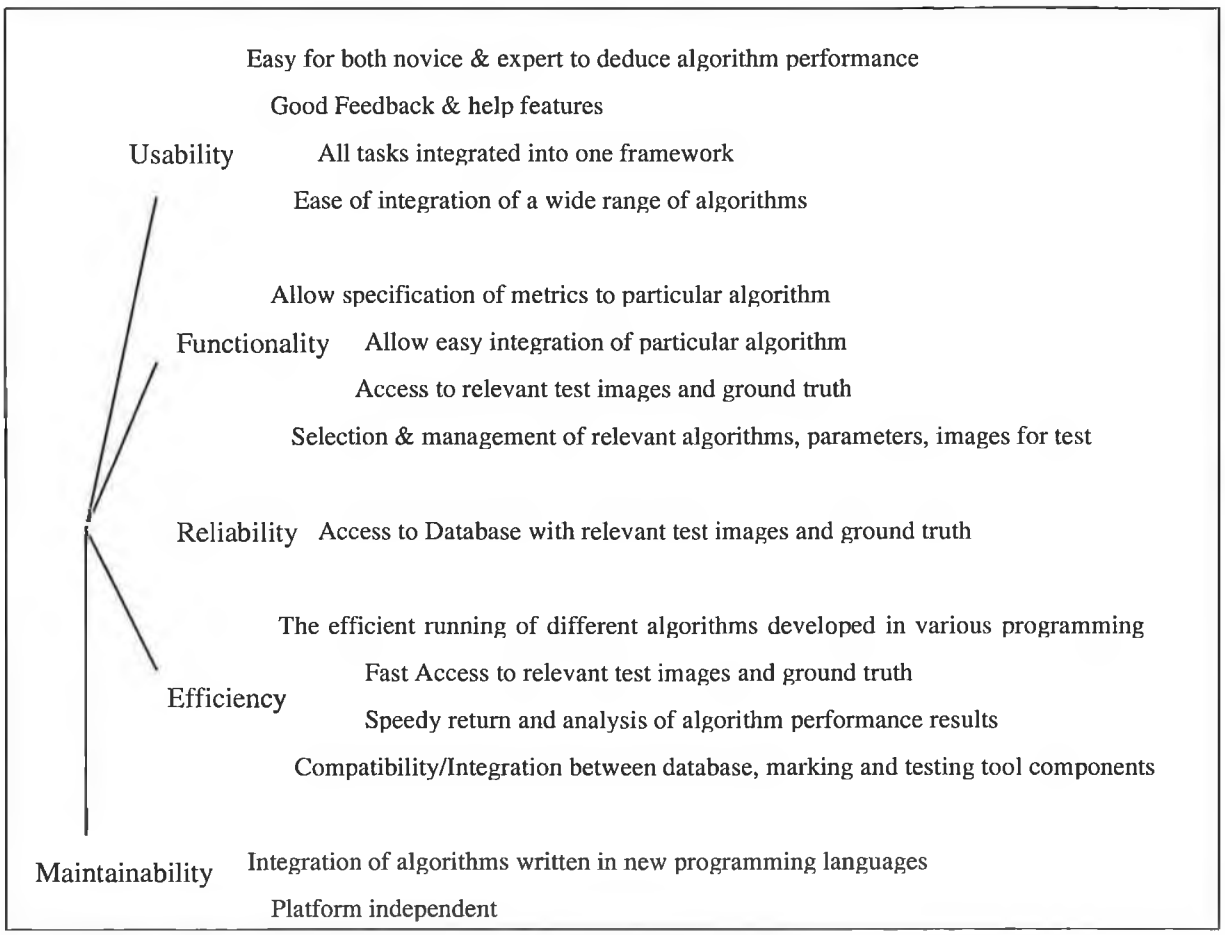


Figure 12: Quality Requirements Tree for Testing Tool

The next section identifies the key non-functional requirements of the testing tool for the two distinct classes of user described above. The algorithm integrator who carries out the

IP algorithm integration process and the algorithm tester who carries out IP algorithm test execution.

4.3.1 ALGORITHM INTEGRATOR

Functionality: No software solution can anticipate the tremendous variety of algorithm types and architectures that may have to be tested into the future. By using a simple, well-defined wrapper between an algorithm and the testing tool, the testing tool should be extensible and customisable to meet various requirements of different algorithms. The differentiation of algorithm integrator and algorithm tester roles mean that algorithm developers will spend more time on understanding algorithm performance, instead of spending time building new testing functionality for each individual algorithm that needs to be tested.

Performance: The integration of algorithms into the test tool should not impede actual algorithm performance in terms of speed and execution time. Algorithm execution time is also one of the main measurements for algorithm performance so the technology used for implementing the algorithm wrapper should be very efficient in terms of performance.

Usability: Usability is concerned with making a software application easy to learn and use. The incorporated wrapper functionality should provide a consistent platform for integrating various algorithms into the testing tool.

"*Usability requirements specify the acceptable level of user performance and satisfaction with the system*" (Preece et al, 1994:385)

As algorithm integration will mainly involve expert users, the testing tool should also emphasise efficiency for proficient users.

Maintainability: Maintainability is the ability to make changes to the testing tool over time. In order to provide comprehensive test functionality into the future, the proposed

testing tool should be designed to anticipate several types of changes in algorithm characteristics. Furthermore the development environment chosen should allow for new component integration easily.

4.3.2 Algorithm Tester

Functionality: A comprehensive evaluation of an algorithm requires it be executed on the most relevant set of test images. Consequently access to a database containing a wide variety of images sorted in a large and easily accessible database is a key component of the test framework (Clark & Clark, 2002:10; Moon et al, 2002:7; Müller et al, 2004). In order to ensure a reasonable level of efficiency the database should provide storage capabilities for a large number of image-test-sets (Kong et al, 2005:111; Sharma & Reilly, 2003). Images by their nature being large in size, the framework should provide a fast means of image transmission between the image database and the testing tool. In addition access to the selected image-test-sets corresponding ground truth is required, as well as a means of specifying relevant metrics of interest to easily deduce assess algorithm performance scores.

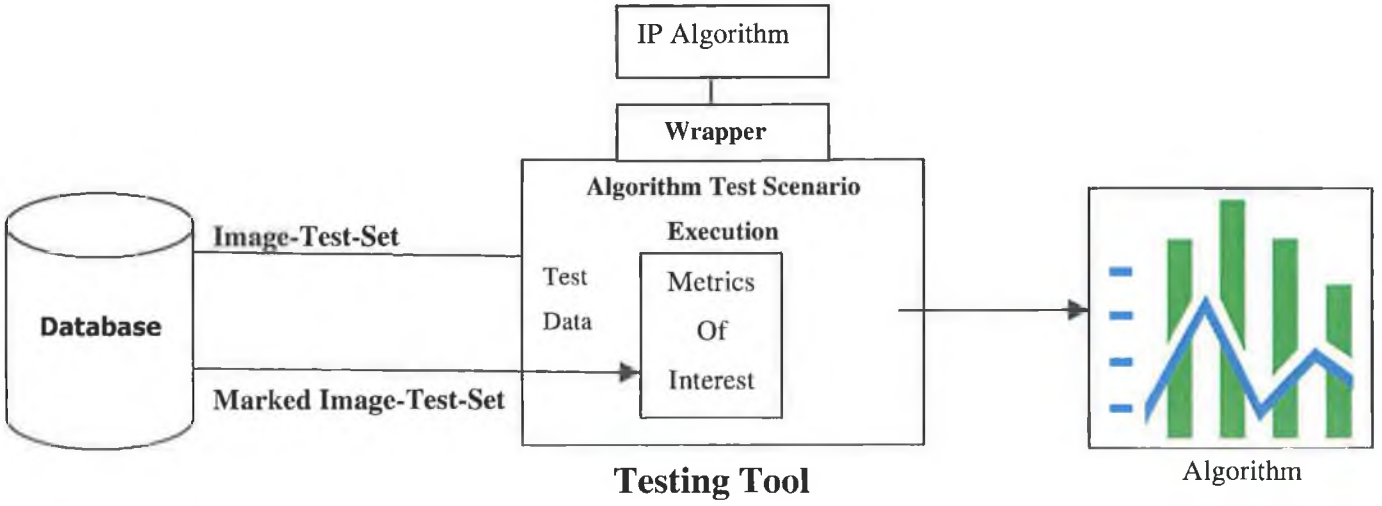


Figure 13: Test Framework Inputs

Usability: A key requirement of all software systems is that they exhibit good usability. Particularly as this software solution proposes to speed up the testing of IP algorithms this requirement becomes even more important.

"*Usability a key concept in HCI is concerned with making systems easy to learn and easy to use. Poorly designed systems can be extremely annoying to users.*" (Preece et al, 1994:14)

The testing tool should provide an easy to use and simple interface that will help speed up the algorithm testing process by allowing algorithm testers to isolate and identify potential algorithm weaknesses quicker and more effectively. With all the requirements for effective algorithm testing incorporated into the one framework, algorithm testers should be able to carry out their task in relatively short time.

"*Algorithm development environments strive to provide the user with an interface that is much closer to mathematical notation and vernacular than are general-purpose programming languages. The idea is that a user should be able to write out the desired computational instructions in a native language that requires relatively little time to master*" (Bovik, 2000:449)

Good feedback and help features should be provided within the testing tool to avoid user errors.

Reliability: Since the system is intended to be available over a network, any number of users may be using the system simultaneously. All users should therefore be guaranteed that they receive the information they requested, be it test results or image-test-sets.

Performance: Obviously a core requirement of the testing tool is efficiency so as to speed up the algorithm testing process. Up to now algorithm developers have had to use different working environments for the different stages in the testing process, namely; importing images, image marking, image categorisation, actual algorithm test execution and performance deduction and analysis. The result of this is a slower and more cumbersome process than is desired in a modern working environment where many incremental algorithm changes and improvements must be applied, each time having to be retested to make sure performance improvements occurred.

To help improve and refine the entire image algorithm testing process the proposed framework should tie together the required components into the one environment:

- Database Tool
- Image Marker Tool
- Testing Tool

Before discussing the different elements of the testing tool's architecture the next section examines the choice of Software Development Lifecycle (SDLC) and technologies employed to develop the entire software solution.

4.4 SELECTION OF SYSTEM DEVELOPMENT LIFE CYCLE AND TECHNOLOGIES

4.4.1 System Development Life Cycle

In the initial project selection phase, a key decision is what System Development Life Cycle (SDLC) to use during the course of the project. The Waterfall model is the most common and classic of SDLC models where each development phase must be completed in its entirety before the next phase can begin. As it is a poor model for complex and object-oriented projects due to the rigidity of the model it was not deemed suitable as the SDLC model for this particular project.

"*In practice, however the development stages overlap and feed information to each other. During design, problems with requirements are identified; during coding, design problems are found; and so on. The software process is not a simple linear model but involves a sequence of iterations of the development activities.*"(Sommerville, 1992:7)

4.4.1.1 Spiral Model

The Spiral life cycle model addresses the shortcomings of the Waterfall model by presenting an incremental development process, in which developers repeatedly assess changing project risks to manage unstable requirements (Nuseibeh, 2001:115). In the

case of this project, because of its iterative and integrated approach to the development process it was the preferred choice of SDLC for a number of reasons:

- Initial project requirements are not very clear.
- New technologies are being deployed.
- It is likely that requirements will change during the course of the project.

More information on the Spiral life cycle model is available in Appendix A.

4.4.2 System Development Technologies

When developing a software application, one of the first questions that must be asked is what programming language to use to develop the application. In choosing a development language a number of factors were taken into consideration including the Integrated Development Environment (IDE) that would be employed to develop the application. The original decision of development language was to use C++ (http://msdn.microsoft.com/visualc/, 2006) because C++ is faster for graphics and image processing. However with recent developments in Java with the powerful and advanced Eclipse IDE (http://www.eclipse.org, 2006) and the Standard Widget Toolkit (SWT) library (http://www.eclipse.org/swt/, 2006), Java revealed itself to be a more preferable choice. It appeared that there were many disadvantages and restrictions that would become more apparent and problematic further down the line when developing in Microsoft C++. Using Java with its advanced Eclipse IDE would ensure that the project is extendable, has the same behaviour across all platforms, and is instantly available for all platforms even when extra features become available. It must also be remembered that the overall IP algorithm test framework is going to be a commercial product, and by not limiting oneself to Microsoft the product can be instantly aimed at a much larger consumer base.

4.4.2.1 Development Language

Java is a sophisticated, standardised, object-oriented programming language (Young, 2002:656) that has emerged as an implementation language of choice for a variety of

software applications. Java is platform independent, and can be run without modification on a broad variety of operating systems (Young, 2002:657). Furthermore, Java incorporates Java Native Interface (JNI), meaning support for imaging libraries written in other programming languages like C and C++ is provided. Especially since a large amount of image processing algorithms are written in high performance programming languages like C++ this is of vital significance.

"*Once an algorithm has been developed and is ready for operational use, it is often implemented in one of the compiled languages such as C, C++, or Fortran for greater efficiency.*" (Bovik, 2000:449)

4.4.2.2 Integrated Development Environment

The choice of Java as development language was predicated on another important reason, the availability of a sophisticated Java development environment, Eclipse. Eclipse is a Java development environment, a tool integration platform, and an open source community all in one (Shavor et al, 2003:6).

"Eclipse has garnered so much support that many industry observers say it is now the key Java-tools player."(Geer, 2005:16)

According to (McAffer & Lemieux, 2005) it is a world-class Java IDE and regularly tops the charts in developer satisfaction and use.

4.4.2.2.1 Eclipse Rich Client Platform

"*It has become one of the most flexible, powerful, and integrated Java development environments.*" (Yang et al, 2005:1)

Although Eclipse was originally built as an integrated development environment for software development, the Eclipse platform can also be used to build client applications (Carlson, 2005; Gruber et al, 2005:289; McAffer & Lemieux, 2005). The Eclipse Rich Client Platform (RCP) is a subset of Eclipse that allows a set of plug-ins to be developed and deployed as a standalone application, independent of the Eclipse development environment. The RCP is composed of a number of components including the workbench which incorporates the editors, views, and perspectives that will make up the

environment that is the overall test framework (Shavor et al, 2003:198). More information on the Eclipse Rich Client Platform is available in Appendix B.

4.4.2.2.2 Eclipse Perspectives

Within the Eclipse workbench, perspectives are the visual containers that hold an appropriate collection of views, editors and actions designed to perform distinct functions.

"*You can tailor the supplied perspectives to make it easier for your users to visualize how you suggest they might want to organize your product's views, editors, and access key actions*" (Shavor et al, 2003:198)

As the perspectives behaviour is task-oriented, the main components of the test framework can be implemented as three separate perspectives:

- A Database Tool perspective.
- An Image Marker Tool perspective.
- A Testing Tool perspective.

4.5 TESTING TOOL DESIGN

Having outlined both the functional and non functional requirements of the proposed testing tool and explained the design of the overall test framework, the next section details the design of the testing tool in depth.

4.5.1 ALGORITHM INTEGRATION ARCHITECTURE

With the wide range of image processing algorithms in existence today (Jaynes et al, 2005:1; Kim et al, 2004; Little et al, 2005:89; Pentland & Choudhury, 2000:51; Yang et al, 2004:2533) developed in a host of different programming languages, it becomes necessary to allow the java developed testing tool to work closely with native code written in other languages. (Bovik, 2000:449) reports that after algorithms have been developed and are ready for use, they are often compiled into a C++ or C imaging library Dynamic-Link Library (DLL) for greater efficiency. An imaging library implements and exports a set of well defined image processing operations (e.g., Red-eye detection library,

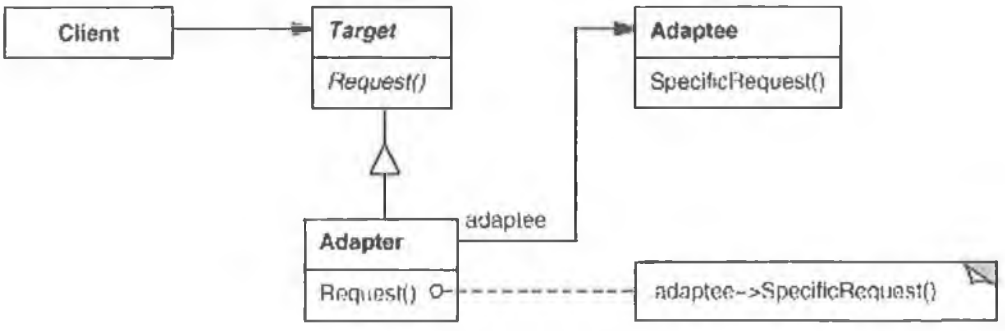
Histogram library). Therefore a means is needed of making native imaging libraries written in C or C++ available to be used inside the Java developed testing tool. In designing a solution for this requirement, the adapter design pattern was utilised.

4.5.1.1 Adapter Design Pattern

The adapter design pattern also known as a wrapper, allows classes work together that couldn't otherwise do so. The intent of the adapter pattern is to convert the interface of an existing class in order to make it compatible with the interface that its client expects (Gamma et al, 1995). In this case, the testing tool will be using third party imaging libraries, and the adapter pattern allows usage of the third party imaging library to be decoupled from the testing tool code. In essence using the adapter pattern adds certain flexibility to the relationship between the testing tool and the algorithm to be tested. All communication between both must go through the wrapper class, meaning that the algorithm can be replaced or updated without the testing tool functionality having to be changed. For instance if a newer version of an imaging library becomes available, having an adaptor interface will allow migration to a newer version very easily, without having to change the testing tool significantly.

"*Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.*" (Gamma et al 1995)

Therefore within the testing tool each individual imaging library interface must have an associated wrapper class that will be responsible for loading the DLL containing the imaging processing algorithm, calling of the actual image processing algorithm routine, returning the results of the call, and also dealing with any error conditions.



As illustrated in Figure 14, because fully automating the application of the adaptor

Figure 14: Adapter Class - Source: Gamma et al, 1995

pattern is non-trivial and needs expert knowledge, the mapping from the new adapter interface to the existing adaptee class should be specified by the programmer. This was one of the main reasons for differentiating the algorithm integrator and algorithm tester roles.

4.5.1.1 Analysis of Implementation Technology: JNI

As the testing tool is been developed in Java, Java Native Interface (JNI) was seen as the most appropriate tool for implementing the wrappers. JNI is a powerful interface that allows the calling of functions, in this case image processing algorithms, written in other languages and usually compiled into a dynamic link library, from Java (Liang, 1999).

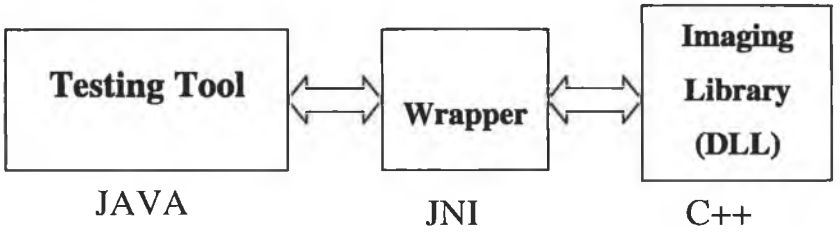


Figure 15: JNI Wrapper

"*A common case of using the JNI is when a system architect wants to benefit from both worlds, implementing communication protocols in Java and computationally expensive algorithmic parts in C++ (the latter are usually compiled into a dynamic library, which is then invoked from the Java code).*" (Gabrilovich & Finkelstein, 2001:1)

The wrapper will provide functionality for:

- Conversion of inputs: Usually image-test-sets and some specified parameters from the format they are stored in the testing tool to a format which the target algorithm can understand.

- Execution of the target algorithm on the image-test-set and any additional parameters.
- Conversion of algorithm outputs to a format that can be used by the testing tool.

Essentially it allows interoperability between programming languages (Liang, 1999; Werbicki & Kremer, 2005). In this case the ability to integrate imaging libraries written in native programming languages such as C and C++ into the testing tool which is to be developed in java. After the algorithm has been integrated into the testing tool the next step is to create a test scenario to evaluate the integrated algorithm's performance.

4.5.2 Test Scenario

Inline with the principles set out in the FERET evaluation methodology (Phillips et al, 2000b), design of the testing tool exploits the direct connection between the algorithm being evaluated, the set of test images and the actual testing protocol. The proposed testing tool allows individual test scenarios to be specified, which incorporate the following components (Clark & Clark, 2002:4; Liu & Dori, 1999:105):

- **Test Data**
- **Ground Truth**
- **Metrics of Interest**

The next section will explain the design of each of these components and give explanations for the choice of implementation technology selected.

4.5.2.1 Test Data

A key requirement of the algorithm testing methodology tool is access to relevant image-test-sets (Jaynes et al, 2005:2). The stored information for image test sets not only consists of actual photographic images, but also image metadata, such as an image description and various other image attributes that may be defined as the need arises. Furthermore as will be explained later, the storage of test scenarios and test results is an

additional requirement of the testing tool. Therefore, data storage is one of the most important design aspects of the overall test framework to be considered. In conjunction with partners on the "Tools & Algorithms to Assist in Automatically Recognising and Deducing Information about People in Consumer Digital Images" project, eXtensible Markup Language (XML) was chosen as the storage mechanism for test related data.

Note: Although relevant image-test-sets and accurate ground truth are essential requirements of the algorithm testing methodology, the development of software tools for the gathering, storage and marking of images is not part of this thesis. This work is carried out by other researchers, working on the "Tools & Algorithms to Assist in Automatically Recognising and Deducing Information about People in Consumer Digital Images" Enterprise Ireland (EI) funded project.

4.5.2.1.1 Analysis of Implementation Technology: XML

XML is a standardised meta-language designed to store, carry and exchange data. XML is completely flexible in how its data can be structured so can be used describe any kind of information including test scenarios and test results (Hunter et al, 2000: 21). As such XML is a perfect fit for the storage and transmission of data associated with the test framework. More information on XML is available in Appendix C.

"*XML simply defines standard ways to manage and exchange complex documents.*" (Orfali et al 1999:625)

4.5.2.1.2 Design Strategy

The client side of the testing tool will therefore be a straightforward system in data storage. By specifying XML schemas for test scenarios and test results, instances of either may be serialised to XML documents and saved to the tests directory on the local machine. An XML schema describes a model for the set of allowed data that is enclosed within an XML document (McLaughlin, 2000). The testing tool then only needs to understand the XML documents constraints described in the schema to utilise data stored in the XML documents (McLaughlin, 2000).

"In many ways, schemas serve as design tools, establishing a framework on which implementations can be built."(Harold & Means, 2004)

As the testing tool will be web-enabled, support will also be provided to upload both test scenarios and test results to a server for central storage. XML provides a means for reducing server load by storing all data on the client for as long as possible and then sending the information to the server in one big XML document. (Hunter et al, 2000: 24) The testing tool will utilise parsing routines to access the stored XML documents. JAXP (Java API for XML Processing) is a standard set of Java APIs for dealing with XML objects (Mordani et al, 2001). The use of parsers enables the testing tool to read the stored XML data and get the relevant information from it.

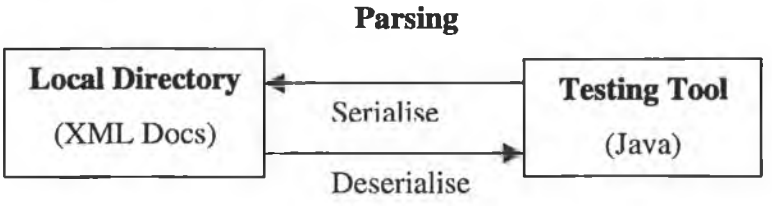


Figure 16: Data Communication and Storage

4.5.2.2 Ground Truth

Ground truth is the corresponding expected output from the algorithm determined by a human and must be compared with the actual results from the algorithm test run to evaluate algorithm performance. As mentioned in previous chapters it usually takes the form of marked features of interest within an image. For instance features of interest in the red eye detection algorithm overview provided in Appendix D are the red eyes. Therefore a means of storing and comparing ground truth with the outputted results from algorithm execution is needed.

Partners working on the "Tools & Algorithms to Assist in Automatically Recognising and Deducing Information about People in Consumer Digital Images" project have developed an XML based language Photographic Feature Markup Language (PFML) used to describe relevant features of interest within images. As defined by the PFML a

feature of interest (red eye) within a photograph, classified as a feature, comprises of a geometry element used to store the marked features co-ordinates. As illustrated in Figure 17, Geometry types defined by the PFML include Point, Ellipse, Rectangle, LineString, LinearRing and Polygon. For instance a point would be used to mark a dust spec on a photograph while a polygon would mark a face.

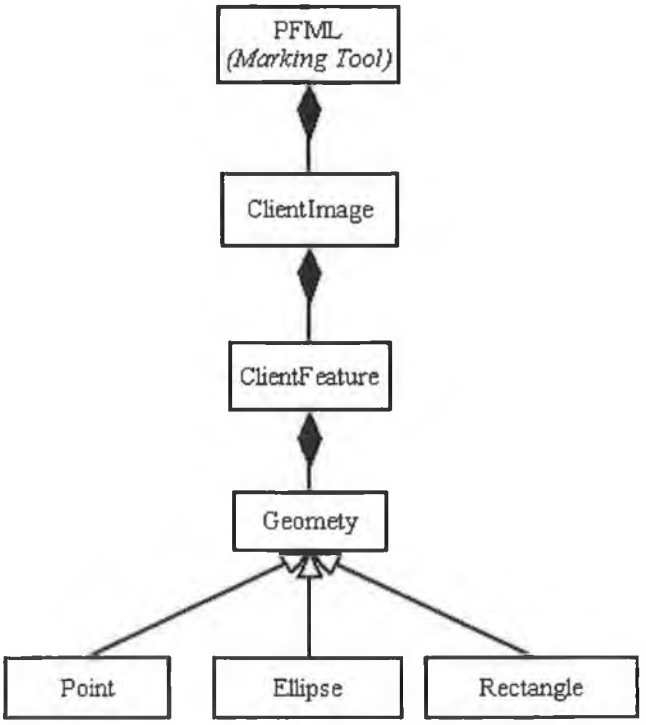


Figure 17: PFML Structure

Essentially algorithm integrators can incorporate these geometry types into their test scripts to compare ground truth with the output from algorithm execution using specific metrics of interest.

4.5.2.3 Metrics of Interest

Performance evaluation is not just finding out whether algorithms perform as expected, it involves the use of objective, usually statistical, measures for comparing the performance of vision algorithms (Courtney & Thacker, 2001; Micheals & Boult, 2000). As can be seen in section 4.5.2.2, the writing of test scripts involves the definition of metrics to

compare outputted results from the algorithm with the corresponding ground truth. The testing tool will allow users to specify metrics of interest to express the difference between the expected output (ground truth) and the actual output (returned results) from the algorithm test (Hua et al, 2004:498, Liu & Dori, 1999).To facilitate this design requirement after algorithms have been integrated into the testing tool, appropriate test scripts incorporating relevant metrics of interest may be wrote to process images in different ways and measure various characteristics of the results of algorithm test scenario execution. The concept of a test script can be defined as the series of image processing or data comparison operations in some programming language, that are executed in order to test various image processing algorithms, and produce test results that may be analysed later. Again it is worth noting with the differentiation of algorithm integrator and algorithm tester roles mentioned earlier that this is essentially a programming task.

Since the testing tool is been developed in java for portability and Graphical User Interface (GUI) component reasons, test scripts will be able to be programmed in Java. Although test scripts written in Java have the advantage of running a lot faster, the disadvantage is that they can only be modified outside the application. The Java class containing the test script must be recompiled and redeployed and the testing tool application must be restarted once again. Obviously a high level of Java expertise is required on the part of the algorithm integrator to complete this task. In designing the testing tool it was thought that integrating functionality to allow test scripts to be written in scripting languages would simplify and speed up the test script writing process. In addition scripting languages have another advantage over Java in that they could be edited from the testing tool and re-run instantly.

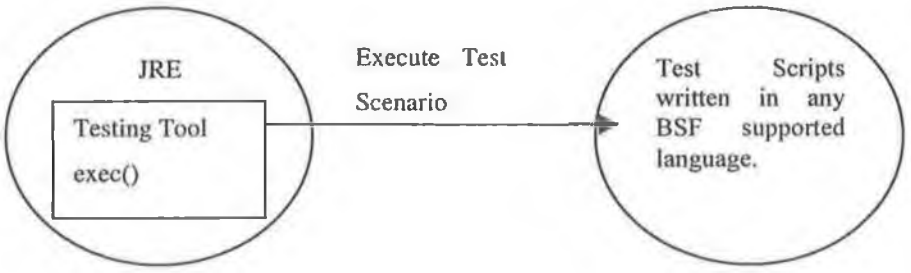
4.5.2.3.1 Analysis of Implementation Technology: Bean Scripting Framework

The Bean Scripting Framework (BSF) (http://jakarta.apache.org/bsf/, 2006) is a set of java classes that enables the use of scripting languages, such as Javascript or Python

(http://www.python.org, 2006), within Java applications and allows the use of java objects and functions within the supported scripting languages. By integrating the BSF into the testing tool, support for the writing of test scripts in several scripting languages is provided including:

- Python provided by way of the Jython engine (http://www.jython.org, 2006).
- Javascript provided by way of the Rhino engine from the Mozilla project (http://www.mozilla.org/rhino/, 2006).
- Ruby provided by way of the JRuby engine (http://jruby.sourceforge.net/, 2006).
- BeanShell (http://www.beanshell.org/, 2006).

Figure 18: Bean Scripting Framework



The advantages of adding BSF architecture to the testing tool for the writing of test scripts include:

- Enabling the testing tool to support a lot of scripting languages easily.
- Enabling "nonprogrammers" to write test scripts

4.5.2.3.1.1 Python

"*Python provides such flexibility, speedy development, and a sense of ease.*" (Bill, 2001)

The easy usage of Python language and the clear structure and simple extensibility means that it is a good choice for a scripting language in the testing tool. Python is a portable, interpreted, object-oriented programming language. The learning and usage of Python is very simple and should allow non-programmers to write test scripts.

4.5.2.3.1.2 Jython

"*Jython is the combination of two programming languages—Java and Python*" (Bill, 2001)

As mentioned earlier BSF provides support for Python via the Jython engine. To incorporate Python test scripts within the testing tool, Jython (http://www.jython.org, 2006) a java implementation of the python programming language will be used. With Jython, Python programs can be wrote that integrate seamlessly with any Java code. And like Python, Jython can be used interactively (Pedroni & Rappin, 2002).

"*Jython, on the other hand, is a powerful complement to existing Java frameworks that blends in transparently.*" (Bill, 2001)

Providing scripting support allows both algorithm integrators and algorithm testers (non-programmers) to write and edit test scripts within the testing tool application and run them immediately. By implementing test scripts in high level scripting languages, the speed of algorithm testing process is also increased. Although not mentioned here, support for Beanshell, Ruby and Javascript scripting languages are also provided within the testing tool.

4.5.2.4 Underlying Architecture Design

Having described the design of the main components of the proposed algorithm testing methodology above, the next section reviews the list of preliminary objects that will be modelled within the system to satisfy their requirements.

4.5.2.4.1 Object Diagram

The steps to model the algorithm testing methodology start by defining a preliminary object-oriented domain analysis diagram as illustrated in Figure 19.

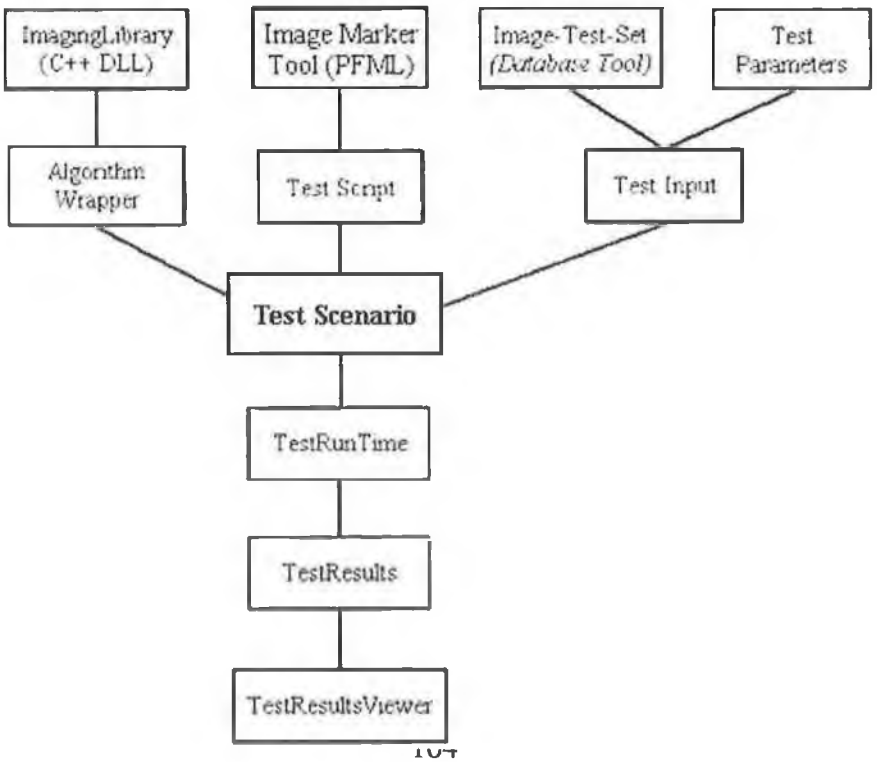


Figure 19: Preliminary Object-Oriented Domain Analysis

First of all, an algorithm is selected for performance evaluation. Since the algorithm is usually encapsulated in a DLL written in a native programming language, an algorithm wrapper based on the specific requirements of the algorithm, allows for the integration of the algorithm into the testing tool. In addition test scripts, which define the series of image processing or data comparison operations that are executed on each image, to produce test results and deduce algorithm performance must be specified. Test scripts may also utilise ground truth, the corresponding marked image-test-set from the image marker tool, stored in PFML format. The algorithm tester first selects a test script to run. Then after, the relevant image-test-set and any additional test parameters are selected, what is known as a test scenario is executed and algorithm performance results based on the test script are returned. Test results are then displayed to the test results view to allow algorithm testers analyse algorithm performance.

4.5.2.4.2 Sequence Diagram

Having defined test scenario execution, the actual running of an algorithm on the image-test-set, as the central task in the algorithm testing methodology the next step is to identify how this will be done. A sequence diagram provided by the Unified Modelling Language (UML) (Booch, 1993), focuses on how a particular function should be undertaken by the architectural elements of the system. Understanding the sequence of a particular task then makes it easier to implement using the appropriate code.

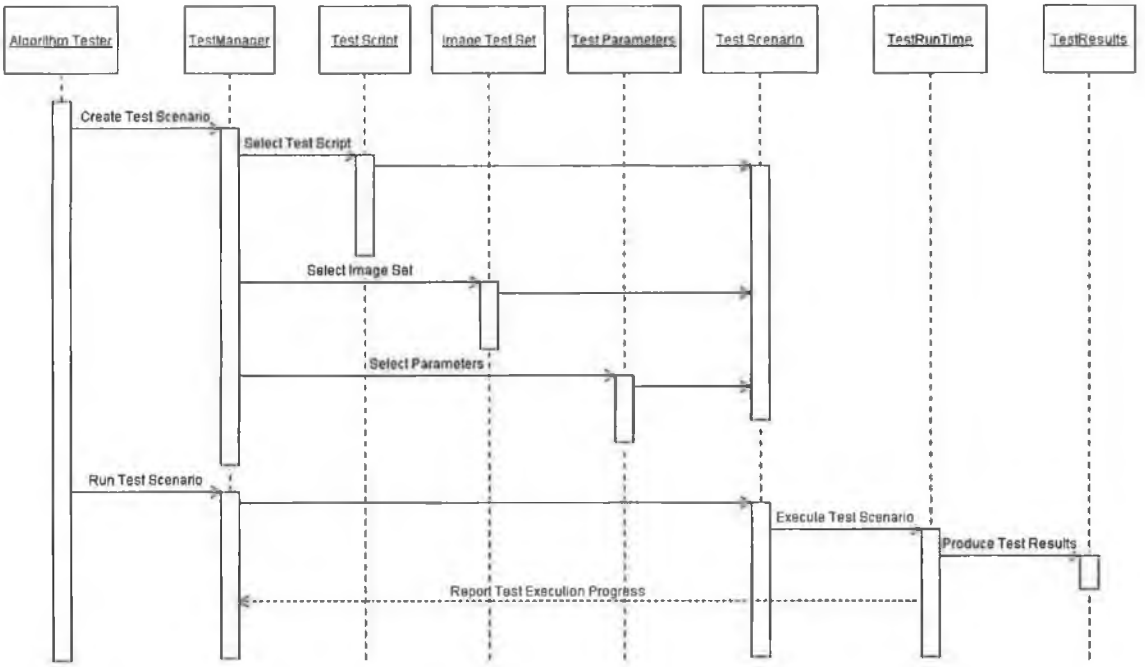


Figure 20: Sequence Diagram

"*During the system design phase, the sequence diagrams are refined to derive the methods and interactions between classes.*" (Kendall & Kendall, 2001:880)

As can be seen in Figure 20, to create a new test scenario the algorithm tester must select a test script, an image test set and any additional test parameters. When the algorithm tester runs the test scenario, it is executed in the test run time module and algorithm performance data is returned as test results. Test results may then be displayed to the appropriate views.

4.5.2.4.3 Design Strategy

4.5.2.4.3.1 Model-View-Controller Architecture

As the acquirement and analysis of algorithm performance data is the basis for development of this system the storage and manipulation of this data should be truly independent of user interface design. The observer design pattern assumes that the object enclosing the data is divided from the objects that present the data, and that these objects will observe changes in that data (Cooper, 1998:177). The Model-View-Controller

(MVC) architecture which will be utilised by the testing tool, is an example of the observer pattern that decouples the user interface from the testing tool functionality and information content (Cooper, 1998:10; Sommerville, 2001:334).

"*Applications developed in Eclipse generally consist of a domain model and a user interface component that display some aspect of the domain model.*" (Shavor et al, 2003:316).

As illustrated in Figure 21, by separating the display of the data from the actual data, the representation on the user's screen can be changed without changing the underlying computational system (Sommerville, 2001:335).

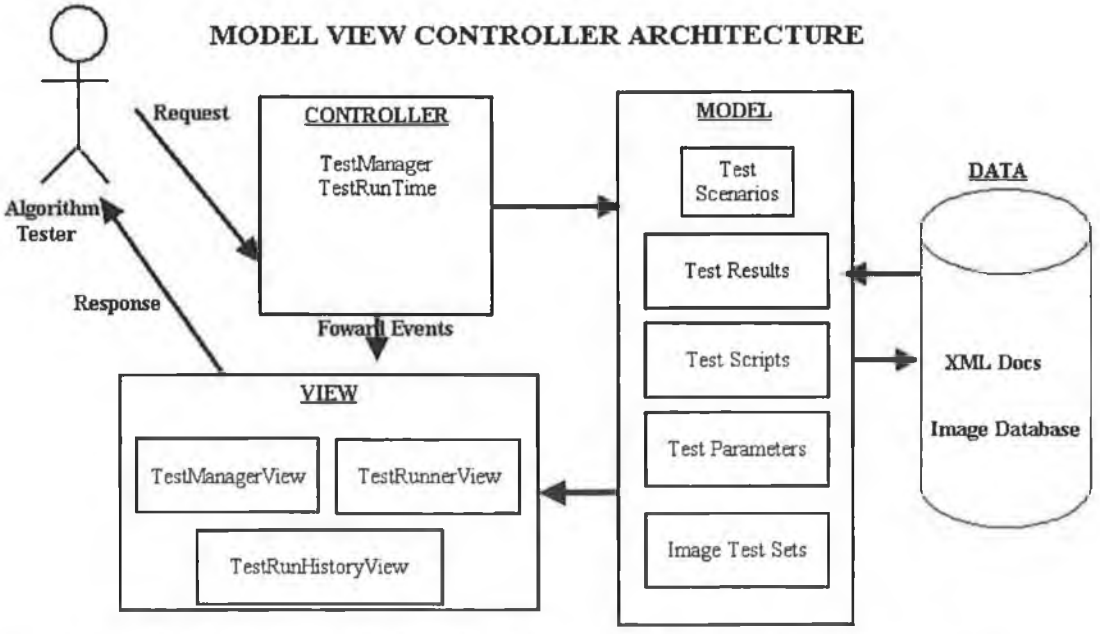


Figure 21: Preliminary Model View Controller Infrastructure Overview

Fig 21 illustrates the main objects in the system. In the model the main entities will be test scenarios and test results. The view is concerned with how information is presented to the user. In this case there will be a "Test Runner" view and a "Test Results" view. The controller containing the Test Manager is mainly concerned with the interactions between the user and the view. Since the MVC separates the View, Model and controller;

system development, design and maintenance is then made easier (Kurniawan, 1999; Shalloway & Trott, 2004).

4.5.2.4.3.2 Test Manager - Singleton Pattern

From Figure 20 and Figure 21, it is clear that a class for managing test scenarios and test results is needed. The singleton design pattern is used in designing classes involved in the central management of resources, in this case test scenarios and test results. A creational design pattern, it ensures a class only has one instance, and provides a global point of access to it. All objects that use an instance of the class will use the same instance.

"*A better solution is to make the class itself responsible for keeping track of its sole instance. The class can ensure that no other instance can be created (by intercepting requests to create new objects), and it can provide a way to access the instance. This is the Singleton pattern.*" (Gamma et al, 1995)

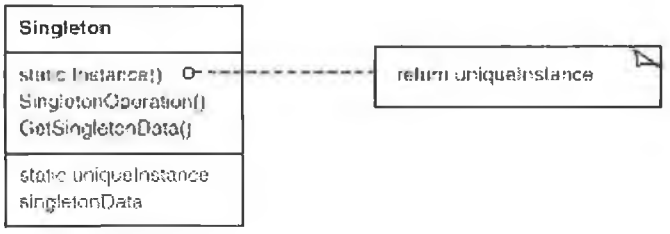


Figure 22: Singleton Pattern - Source: Gamma et al, 1995

Its main advantage is it controls access to the stole instance of the test manager.

4.5.3 Graphical User Interface (GUI) Design Process

Good Graphical User Interface (GUI) design is critical to the success of any software application (Sommerville, 2001:328). An interface that is difficult to use will result in a high level of user errors. If for instance performance information is presented in a confusing or misleading way, users may be unable to deduce algorithm performance. Therefore, during the GUI design stage, screen design prototypes were utilised to design and refine the layout of the user interface.

"*Prototyping is an appropriate way to communicate design information to users.*" (Preece et al, 1994:563)

Prototypes were quick and inexpensive to use and they provided invaluable insights into UI design. A preliminary use case diagram (Figure 23) depicts the main functionality of the testing tool purporting to the role of the algorithm tester.

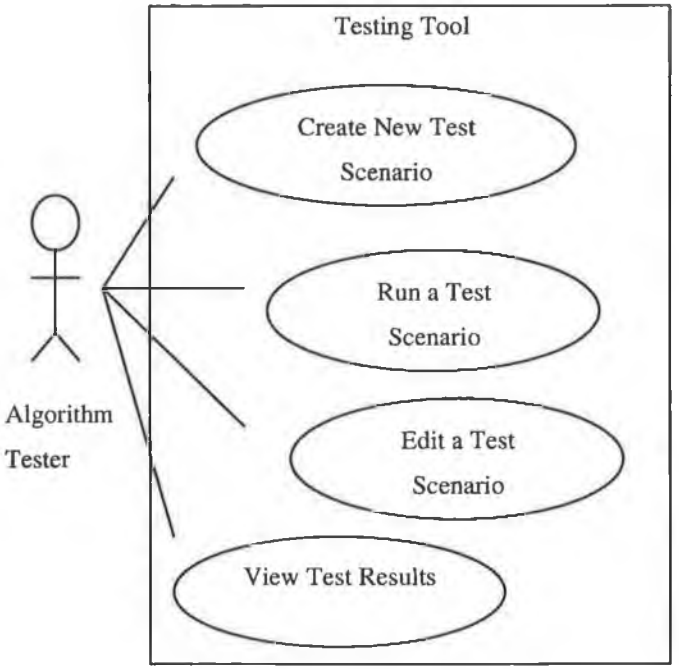


Figure 23: Use Case Diagram for Testing Tool

For the development of a reliable and successful GUI, two questions had to be answered:

- What information should be displayed?
- How should it be displayed?

As examined in the use case displayed in Figure 23, the main information to be displayed in the GUI is the list of test scenarios which can be run, information on the running process and the actual tests results. Therefore it was initially decided to divide the testing tool UI up into three display blocks or views.

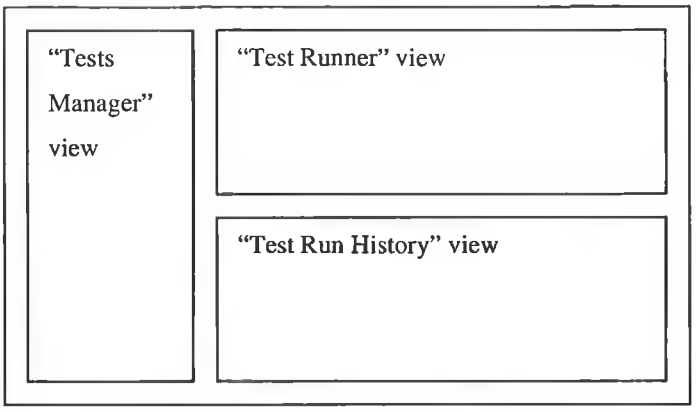


Figure 24: Preliminary GUI Design

As illustrated in Figure 24, the "Test Manager" view will display a simple tree hierarchy of the test scenarios which can be run. When a test is run, the "Test Runner" view will display information on the running status and performance results of the test scenario while the "Test Run History" view will display previous test run results for the currently selected test scenario.

"*Developing prototypes is an integral part of iterative user-centred design because it enables designers to try out their ideas with users and to gather feedback.*"(Preece et al, 1994:537)

As a result of early design evaluation it was decided to revise the initial screen design prototype (Figure 24) and place the "Test Runner" view and "Test Run History" view into two separate windows (Figure 25). The main reason for this design refinement was that information displayed in both views was large and both constituted more space.

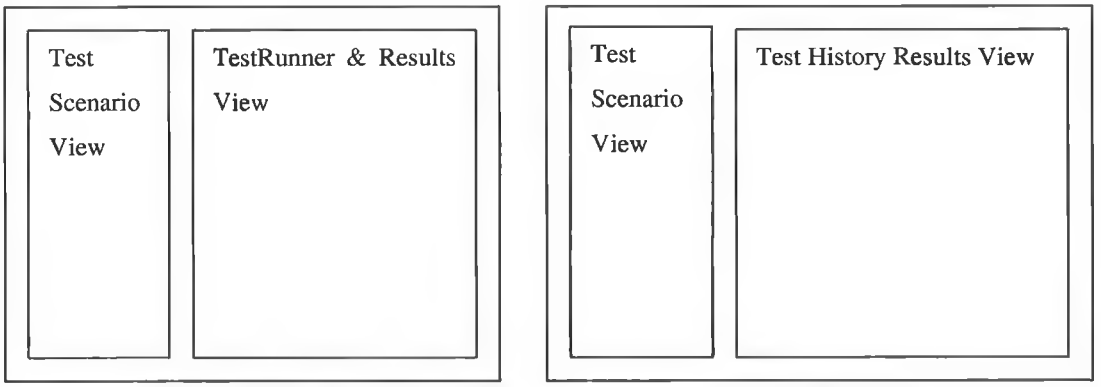


Figure 25: Revised GUI Design

As mentioned in section 4.5.2.4.3, design of the testing tool will implement the Model View Controller (MVC) architecture. The MVC architecture of separating the model from the user interface is a fundamental pattern applied to the classes involved in building views in Eclipse. Implementing the above blocks as views illustrated in Figure 25 within the eclipse workbench will be further discussed in the implementation chapter.

4.5.3.1 Analysis of Implementation Technology: Standard Widget Toolkit

Eclipse is based on an alternative Java graphics toolkit called the Standard Widget Toolkit (SWT) (Knudsen & Niemeyer, 2005); a widget toolkit that provides a set of portable APIs that allows developers to build GUIs easily (Hatton, 2005:1). SWT has tight integration with the underlying native Operating System GUI platform, which enables efficient and portable access to the native GUI facilities of the Operating System (McAffer & Lemieux, 2005).

"It lets developers build portable applications that directly access the user-interface facilities of the operating systems on which they are implemented."(Geer, 2005:16)

Therefore SWT applications can have the identical look and feel to applications developed entirely in native code on a particular platform (Hatton, 2005:2). The decision to use SWT as opposed to the Abstract Windowing Toolkit (AWT) and the Java foundation classes (SWING) for the development of the testing tool's graphical components essentially came down to performance issues. SWT has improved performance and memory consumption as opposed to Swing (Hatton, 2005:1; Knudsen & Niemeyer, 2005; Pluta, 2004:6).

4.5.3.1.1 JFace

JFace is a platform-independent user interface API implemented using SWT that provides classes for handling common UI programming tasks. Essentially the JFace UI framework provides higher-level application constructs for supporting data viewers, dialogs, wizards, and actions components.

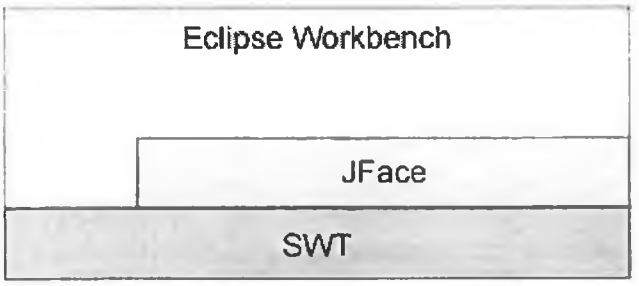


Figure 26: Eclipse Workbench, JFace, and SWT - Source:
http://www-128.ibm.com/developerworks/library/os-ecgui1/

As will become clear in the implementation chapter, extensive use was made of both SWT and JFace classes during development of the testing tool.

4.6 SYSTEM DESIGN EVALUATION

This section explains how the chosen design satisfies each of the initial requirements mentioned in previous chapter. Based on the various technologies decided for system implementation, the system architecture of the testing tool is depicted below.

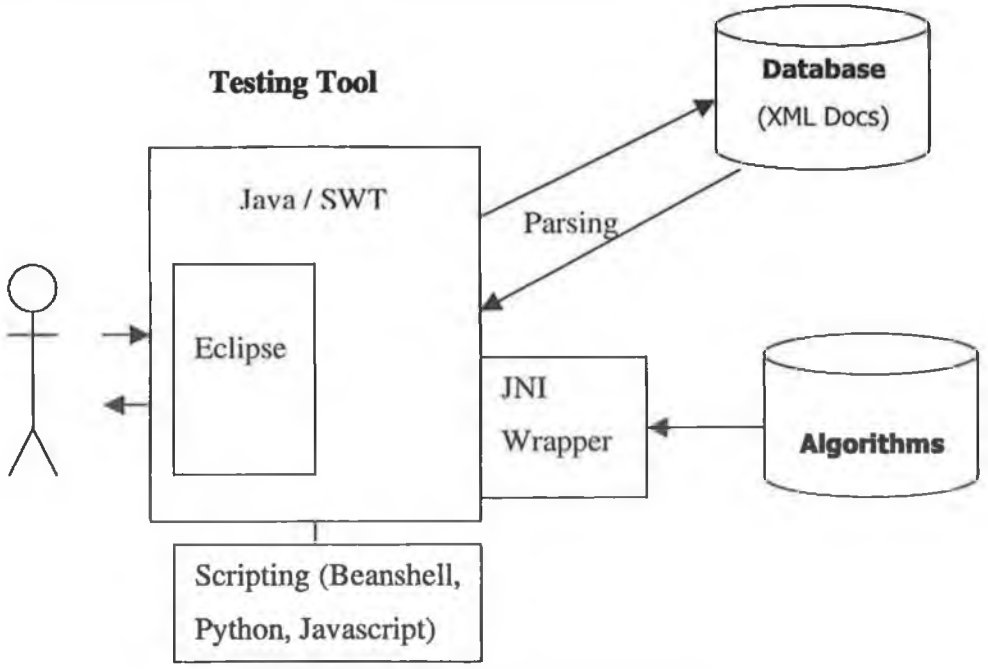


Figure 27: System Architecture

The core problem addressed by the test framework is how to integrate various image processing algorithms for performance evaluation to support a semi-automatic testing processing. By incorporating JNI wrapper functionality the test framework is extensible and efficient so as to allow the analysis and performance assessment of a wide range of image processing algorithms. This allows algorithm testers spend more time on understanding algorithm performance instead of spending time trying to implement algorithms for testing purposes. By utilising the Rich Client Platform of the Eclipse IDE (Gruber et al, 2005:289), the testing tool, image marker tool and database tool can all be integrated into the one application meaning that algorithm performance is carried out quickly and easily, thus greatly reducing testing time (Clark & Clark, 2002:12). The integrated database tool provides easy access to a comprehensive set of test images (Jaynes et al, 2005:2; Kong et al, 2005:111; Moon et al, 2002:7; Yang et al, 2002:52) and the integrated image marker tool allows for ground truth to be acquired quickly and accurately (Micheals & Boult, 2001:152; Müller et al, 2004).

The generic nature of the testing tool allows for test scripts to be written incorporating the specified metrics of interest that will be applied to algorithm execution output based on the algorithm tester requirements. In meeting the specified non-functional requirements of good usability and high performance, test scripts can either written in java or in one of the supported scripting languages by a non-programmer. Furthermore the test scenario architecture facilitates the decoupling of the algorithm from its operating conditions resulting in a more comprehensive algorithm assessment (Moon et al, 2002:1; Ojala et al, 2002:705; Phillips et al, 2000b). Finally by utilising SWT and JFace to construct the testing tool GUI and the use of XML to store the related algorithm test scenario data, the testing tool allows for performance data to be displayed graphically in a suitable format facilitating fast and effective interpretation by both novice and expert users (Sharma & Reilly, 2003).

4.7 CHAPTER SUMMARY

This chapter has drawn up a blueprint for system implementation called the design. Firstly, the overall algorithm test framework was described and the classification of the two distinct categories of testing tool user was explained. The key non-functional requirements based on the two categorises of user for the testing tool were then identified. After the selection of system development life cycle and technologies were explained, the following sections outlined the main design aspects of the testing tool. Firstly design of the testing tool algorithm integration architecture was explained. The subsequent section then described the design of the components that make up a test scenario and the underlying test execution architecture, while the penultimate section detailed the GUI design process. The concluding section then explained how the chosen design satisfies each of the initial requirements defined in the previous chapter.

CHAPTER 5: IMPLEMENTATION

Implementation

5.1 INTRODUCTION

This chapter will focus on describing some of the important implementation techniques which were used during the development of the testing tool. The chapter commences by giving an overview of the overall algorithm test framework. With a similar approach to the design stage in the previous chapter taken, Section 5.3.1 describes the implementation of the testing tool algorithm integration architecture. Section 5.3.2 goes on to describe the development of the components that make up a test scenario and the underlying test execution architecture, in particular explaining the process of writing test scripts incorporating the relevant metrics of interest to analyse algorithm performance. Section 5.4 provides an overview of the overall testing application UI, while finally Section 5.5 details the GUI of the testing tool. In addition code snippets are provided throughout this chapter to give a deeper understanding of how the testing tool really works.

5.2 IP ALGORITHM TEST FRAMEWORK

5.2.1 FRAMEWORK OVERVIEW

As mentioned in the previous chapter the overall IP algorithm test framework is divided into three separate tools, which handle its various functions:

- The **Image Database Tool** will provide access to a comprehensive set of relevant test images that will allow algorithms be tested on the most applicable image-test-sets for a complete evaluation.
- The **Image Marker Tool** will allow for the accurate and rapid marking or annotation of features of interest within images to be used as ground truth within the testing process.
- The **Testing Tool** allows various algorithms to be plugged into testing tool for performance evaluation. After appropriate metrics of interest have been specified the algorithms can then be run on the most pertinent image-test-sets to deduce algorithm performance.

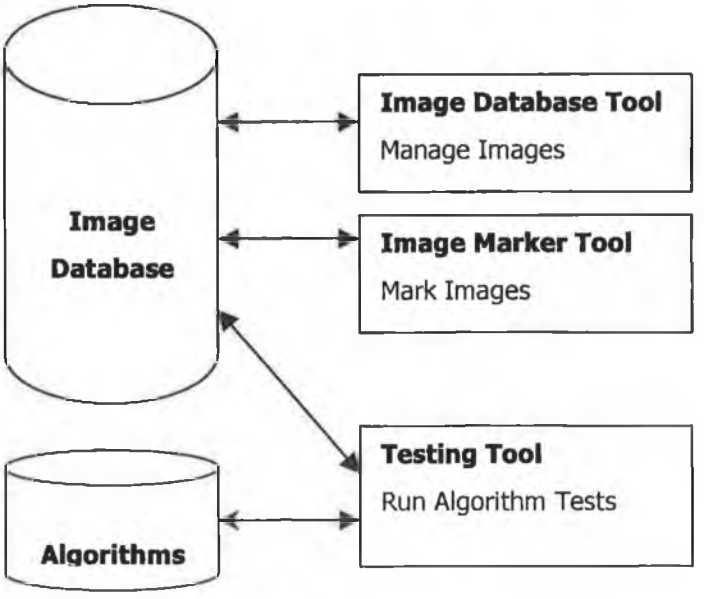


Figure 28: Test Framework Architecture

5.3 TESTING TOOL IMPLEMENTATION

This thesis is principally concerned with the design and development of the testing tool so the next section will describe its implementation. With a similar approach to the design

process taken in previous chapter, the next section first describes the implementation of the testing tool algorithm integration architecture

5.3.1 Algorithm Integration

As identified in previous chapters, one of the most important requirements of the algorithm test framework is extensibility. Unlike a lot of current image processing algorithm testing environments, the testing tool emphasises easy extension by use of library wrappers. In essence the wrapper allows performance evaluation of specific algorithms designed to do specific tasks (Blackburn, 2001; Courtney & Thacker, 2001:3; Phillips et al, 2000a).

(Bovik, 2000:449) reports most IP algorithms are compiled into a C++ or C imaging library DLL for greater efficiency. An imaging library implements and exports a set of well defined image processing operations (e.g., Red-eye detection library, Histogram library). For integration into the testing tool, all imaging libraries must implement a common interface called ImagingTool. The algorithm test scenario accesses and uses any of the available imaging libraries through this interface. For example image input parameters are passed in to the Java library wrapper class as a Java ImageBuffer object, which is a thin wrapper for the IO_Img C data-structure from the ImageIO library. This library and data structure is used throughout the application by algorithm test scenarios to load, convert and save images. The ImagingTool source code is available in Appendix E.1.

As mentioned in the design chapter, the testing tool uses Java Native Interface (JNI), a technology built into Java to call image processing algorithm routines written in C++. The task of calling routines written in C++ from Java comes under the heading of implementing and using "native" methods. Since the testing tool is written in Java, to make a native imaging library written in C or C++ available to be used within the testing tool, the native imaging library must first be wrapped in an adapter layer. This adapter layer is generally referred to as a "library wrapper". See Figure 29.

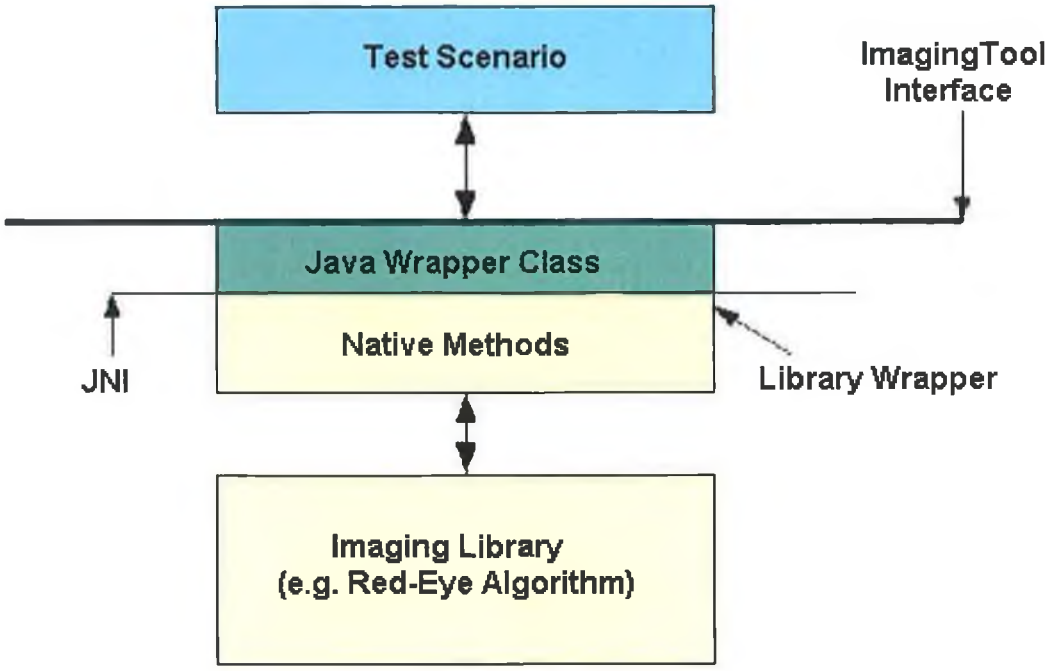


Figure 29: Imaging Library Wrapper

Essentially once the imaging library is wrapped, the testing tool can call the Java wrapper class, which in turn calls the native method, which finally calls the standard C++ imaging-library function to execute the image processing algorithm. The following section outlines the steps involved in building a java library wrapper. An example of integrating an image processing algorithm written in Python into the testing tool is provided to better explain the process.

5.3.1.1 Steps in Building a Java Library Wrapper

Building a Java library wrapper for a native library means creating a Java class that supports the same logical operations as the ones defined in the imaging library. The Java class that implements the wrapper will have a number of JNI native methods written in C or C++ that perform the following operations:

1. Convert the Java input parameters into C/C++ data structures.
2. Call the relevant functions in the imaging library to implement the target operation.

3. Convert the C/C++ result data into Java objects and return this result.

4. In case of errors, convert the C/C++ errors into Java exceptions and throw these exceptions.

Therefore building a Java library wrapper involves the following steps:

1. Evaluate the exported interface of the target imaging library, and define the equivalent Java API operations so that these match the ImagingTool API.

2. Create the Java class that implements the ImagingTool interface and can perform the operations defined at the previous step; the class will contain one or more native methods that perform the required operations using the imaging library.

3. Generate the C header file for the native methods using javah. Sun (http://java.com/en/about/, 2006) provides a utility "javah" which takes the compiled java class that implements the ImagingTool and generates a header file from it. Inside this header file is a C-style declaration for each native method.

4. Write the C++ implementation of native methods using C/C++ IDE and build the resulting wrapper DLL.

5. Deploy the new wrapper by making its DLL available to the testing tool at run time. The imaging library must similarly be made available.

As explained, JNI is used when existing imaging libraries implemented in high performance languages other than Java need to be integrated into the testing tool for performance analysis. Rather than continue discussing the process of writing wrappers in an abstract way, the next section explains the process of writing a wrapper for a specific example, HistogramLib by following the steps described in the previous section.

5.3.1.2 Example: Building a Library Wrapper for a Histogram Wrapper

The Histogram algorithm is one that detects contrast in images i.e. the variation in colour. Not all photographs taken are perfectly exposed and most general purpose algorithms should work as well on over exposed images as on under exposed images.

"*Statistical measures can be extracted from a digital image to quantify the image quality.*"(Shirvaikar, 2004)

The histogram algorithm is used to filter out unsuitable images, those that are over exposed or underexposed from image-test-sets before an algorithm is run on them. For instance if the image-test-set is over exposed, too bright, it may be impossible for the red eye detection algorithm to correctly detect red eyes within the image-test-set. Essentially it measures the colour distribution of an image finding out what images are over exposed or underexposed. The algorithm is typical of many image processing algorithms; an input image is passed to the algorithm and the resulting output is a data structure.

Step 1: Evaluate the exported interface of the target imaging library.

The exported interface of the target imaging library in this case the HistogramLib is first evaluated, and the equivalent Java API operations defined so that these match the ImagingTool API. The target library has a single exported function call GetHistogram that is used to get a simple histogram from an input image; the user may also specify a rectangle that marks a region in the image, and the histogram will be calculated only from that region. The function takes an image buffer and a rectangle as input parameters. The output parameter is an int[256] array that contains the histogram data. Image input parameters can be passed in to the Java library wrapper class as a Java ImageBuffer object, which is a thin wrapper for the IO_Img C data-structure from the ImageIO library. This library and data structure is used throughout the application to load, convert and save images. Therefore, it is clear that the wrapper must contain a single operation that takes the image buffer and region details as input data and will return an int array that contains the histogram information. The histogram.h source code is available in Appendix E.2.

Step 2: Create the Java class that implements the ImagingTool interface and can perform the operations defined in Step 1.

The Java wrapper class named HistogramWrapper that implements the ImagingTool interface is created which performs the operations defined at the previous step; the class will contain one or more native methods that perform the required operations using the imaging library. Figure 30 illustrates a class diagram of the HistogramWrapper and ImageIOTool classes.

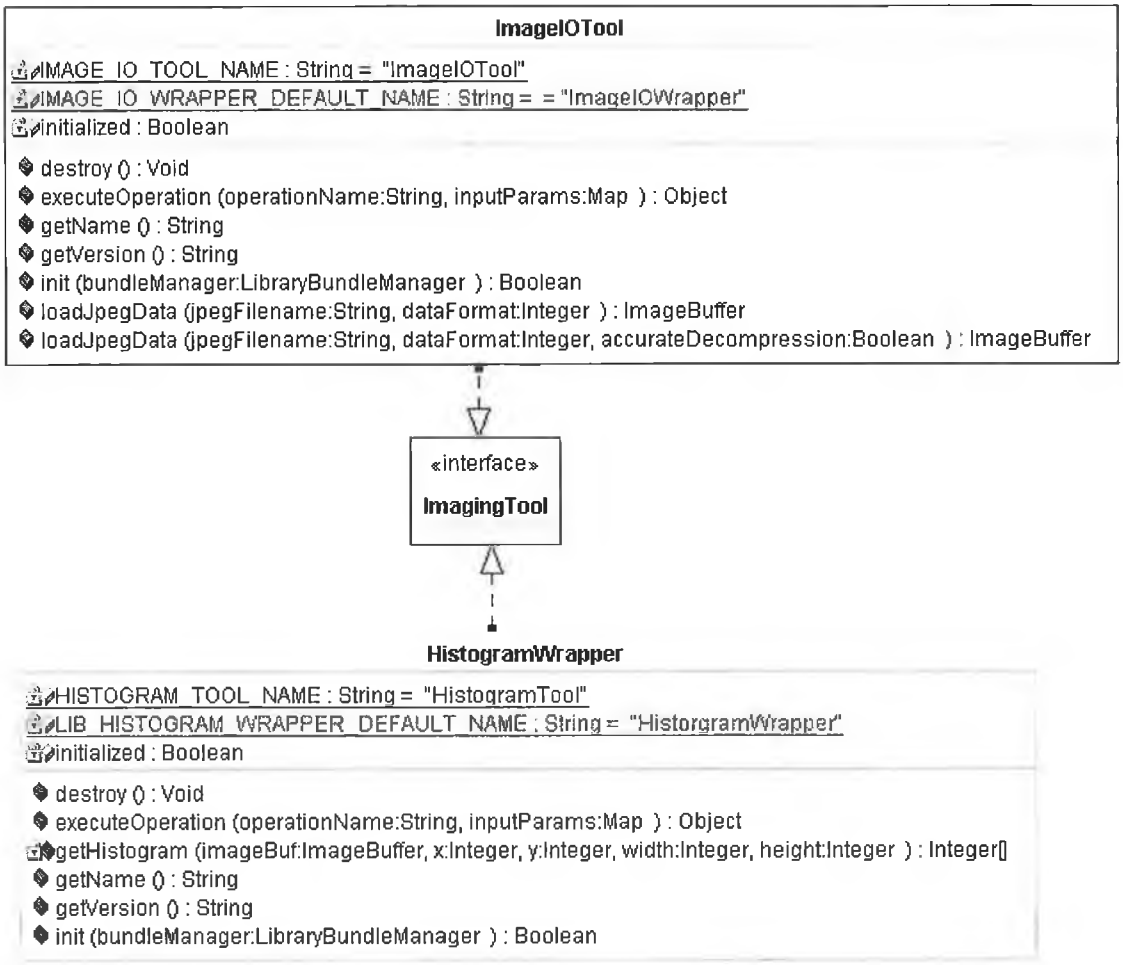


Figure 30: HistogramWrapper Overview

Step 3: Generate the C header file for the native methods using javah.

Now the C header file for the native methods must be generated. Generating the header file is pretty straightforward, with use of the javah.exe command line tool that is part the standard JDK distribution (http://java.com/en/about/, 2006). The following command is executed: `D:\j2sdk1.4.2_06\bin\javah.exe -jni HistogramWrapper`

This generates the header file for the native part of the wrapper. For this to work the HistogramWrapper class must be in the classpath. The generated header file is named HistogramWrapper.h, and it's source code is available in Appendix E.3.

Step 4: Write the C++ implementation of native methods using C/C++IDE and build the resulting wrapper DLL.

This step involves implementing the native methods using a C/C++ IDE, and building the resulting DLL. To do this a Visual Studio DLL project (http://msdn.microsoft.com/visualc/, 2006) is created and the HistogramWrapper.h header file is added to the project. The HistogramWrapper.cpp file is then created that will implement the native method defined in the header file.

The getHistogram routine has a simple API:

- Pass in an image and a rectangle as input parameters which are converted by the JNI native method to the relevant C++ format.
- Returns an int[256] array that contains the histogram data

Once the code has been written and all the errors are fixed, the code can be compiled and the wrapper DLL built. The HistogramWrapper.cpp source code is included in Appendix E.4.

Step 5: Deploy the new wrapper by making its DLL available to the testing tool at run time.

Now that the Java wrapper class, the wrapper DLL, and the imaging library DLL are available, these must be deployed inside the testing tool. This involves copying the DLLs into predefined folders, adding the Java wrapper class to the classpath of the testing tool and registering it to be available to be used by the testing tool. With wrapper support added for the relevant imaging libraries, work can be carried out on creating and running specific test scenarios to test the algorithms performance.

5.3.2 Test Scenario

The proposed IP algorithm testing methodology allows individual algorithm test scenarios to be specified based on algorithm developer's requirements. Algorithm test scenarios are the central point of the overall test framework, around which all other software components are built. An algorithm test scenario allows the definition of what parameters the algorithm takes as input, usually the image-test-set, what output the algorithm returns and how this is compared with ground truth to deduce performance scores (Clark & Clark, 2002:4; Föerstner, 1996:12; Liu & Dori, 1999:105). As illustrated in Figure 31, an algorithm test scenario has the following components:

- Test Input Data:
  - Image Test Set - The set of images which the test script is executed on.
  - Test Parameters – Set of Input parameters that specify additional input used by test script.
- Test Script: Specifies the set of instructions to be executed on each image and produces a set of data fields of various measurements and metrics, which are the results of the test run for the particular image.

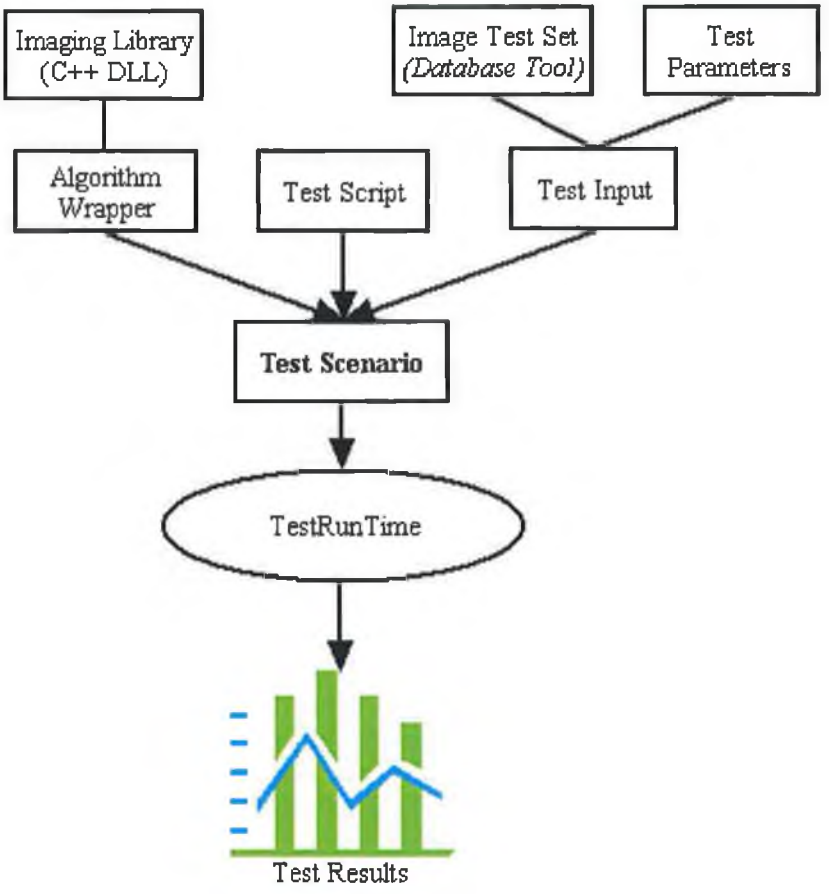


Figure 31: Test Scenario Overview

Note: Although relevant image-test-sets and accurate ground truth are essential requirements of the algorithm testing methodology, the development of software tools for the gathering, storage and marking of images is not part of this thesis. This work is carried out by other researchers, working on the "Tools & Algorithms to Assist in Automatically Recognising and Deducing Information about People in Consumer Digital Images" Enterprise Ireland (EI) funded project. Therefore the next section focuses on describing the implementation of test script functionality.

5.3.2.1 Test Scripts

As mentioned in the design chapter to allow the relevant characteristics of various algorithms to be assessed, test scripts must be written incorporating appropriate metrics of interest. Test scripts specify a set of instructions that are executed on each input image, to produce a set of data fields of various measurements and metrics. Apart from programming test scripts in java, the testing tool supports the writing of test scripts in various scripting languages through the use of the Bean Scripting Framework (BSF). The two core elements of BSF's architecture are the BSFManager and the BSFEngine. The BSFManager is a common interface to scripting languages: while the BSFEngine interface provides a common interface for BSF to cooperate with a scripting language.

After the relevant scripting language is registering with the BSF manager, test scripts can then be wrote and stored in a file directory on the algorithm tester's computer. Given the name of the test script file, BSFManager can then tell which scripting language to use to execute the test script. Before evaluating and executing scripts, objects (or an entire object model) may be mapped into the BSF manager. All objects that are mapped in the BSF manager are available to the scripts. One way to map objects into the BSF manager is to call the declareBean method. As illustrated in Figure 32, the declareBean method takes three arguments: the script variable name of the bean (object), the instance and the class of the instance.

```
this.bsfManager.declareBean("ImageMarkup",ImageMarkup.class,Class.class);
this.bsfManager.declareBean("TestParam",TestParam.class,Class.class );
this.bsfManager.declareBean("TestResult",TestResult.class,Class.class);
this.bsfManager.declareBean("Geometry",Geometry.class,Class.class );
```

Figure 32: Mapping java objects into BSF Manager

5.3.2.1.1 Writing the Test Script for Execution

As mentioned in previous chapters the proposed algorithm testing methodology should allow users to stipulate metrics appropriate to the particular class of algorithm being tested. Test scripts specify a set of instructions encompassing various measurements and metrics

that are to be executed on each image, the results of which characterise algorithm performance. To create a test script for the histogram imaging tool, the python script illustrated in Figure 33 was wrote. The test script calculates the percentage of the dark, light and other pixels in a set of images.

```
blackPixelLimit = 80
whitePixelLimit = 200
imageBuffer = imageIOTool.loadJpegData( image.getFullResURL(), format )
totalPixelCount=imageBuffer.getPixelWidth()*imageBuffer.getPixelHeight()
params = HashMap()
params.put( "imageBuffer", imageBuffer )
histoTool = testContext.getImagingTool( "HistogramTool", None )
histogram = histoTool.executeOperation( "getHistogram", params )
result.addResultField( "Image ID", image.getId() )
counter = 0
for i in range( blackPixelLimit ):
counter += histogram[ i ]
result.addResultField("DarkPixel%",float(counter)/totalPixelCount)*100)
counter = 0
for i in range( whitePixelLimit, 255 ):
counter += histogram[ i ]
result.addResultField("LightPixel%",(float(counter)/totalPixelCount)*100)
counter = 0
for i in range( blackPixelLimit+1, whitePixelLimit-1 ):
counter += histogram[ i ]
result.addResultField("OtherPixel%",(float(counter)/totalPixelCount)*100)
```

Figure 33: Histogram Test Script

On execution the test script will produce a table of results, with a row for each image from the input image set. Each row contains the image-id, and the calculated dark, light and other pixel percentage values as columns. The test scripts provide for customisable results display. Performance data may be displayed graphically in a suitable format of columns and rows facilitating fast and effective interpretation by both novice and expert users (Sharma & Reilly, 2003). The Histogram test is not concerned with ground truth, so access to the Photographic Feature Markup Language (PFML) is not required here. Metrics of interest are simply used to calculate the percentage of the dark, light and other pixels in the set of images.

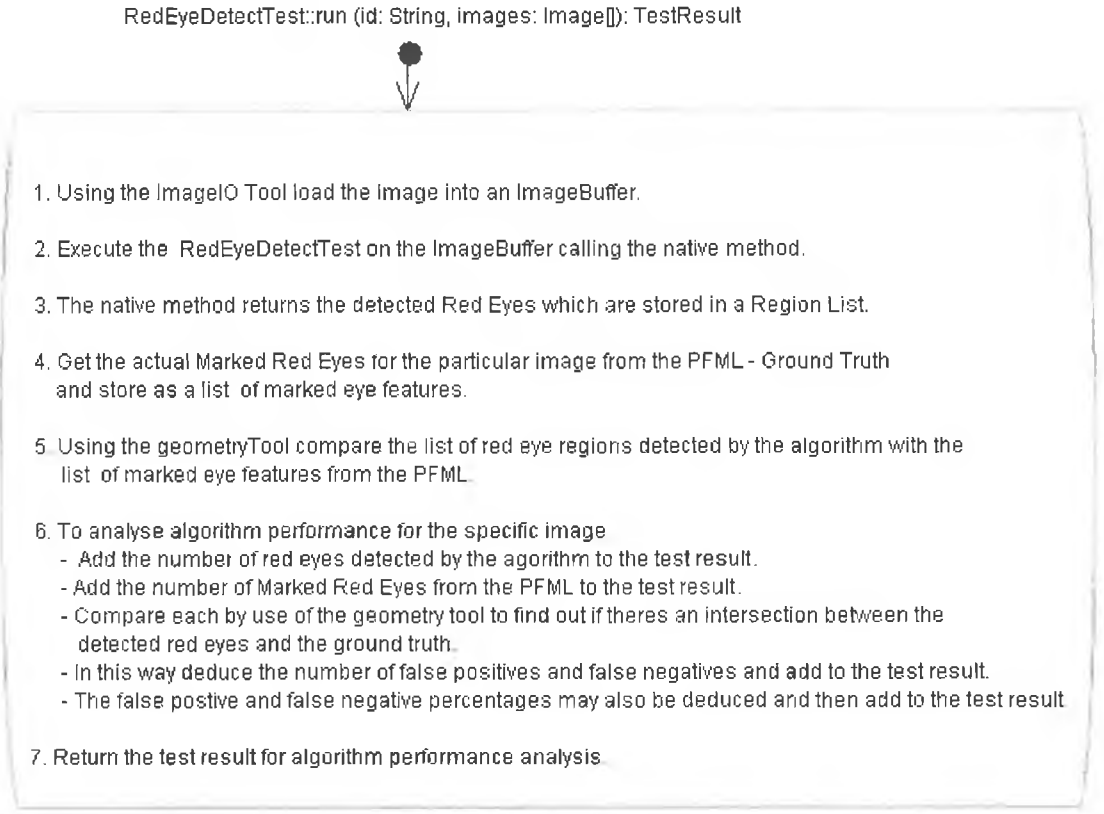


Figure 34: Activity Diagram for RedEyeDetectionTest Run Method

An activity diagram is presented in Figure 34, explains how ground truth would be used in a test script to evaluate the performance of a red eye algorithm. The Photographic Feature Markup Language (PFML) is used to store and describe the relevant features of interest marked within images. After the application's scriptable objects have been mapped into the manager and the test script written, the script is then execution ready to evaluate algorithm performance as illustrated in Figure 35.

```
this.bsfManager.exec("jython",this.scriptFilename,0,0,this.cachedScript);
```

Figure 35: Executing a Python Script

5.3.2.2 Test Manager

Obviously with the testing tool allowing the analysis and performance assessment of a wide range of image processing algorithms multiple test scenarios will be created and executed. The TestManager class is developed as a singleton class and is responsible for managing algorithm test scenarios and test results. It loads all test scenarios and test results stored in XML format from the test store directory. The TestManager source code is available in Appendix E.5.

5.3.2.2.1 Test Scenario Storage

An XML Schema, a World Wide Web Consortium (W3C) XML language is used for describing the contents of a stored test scenario instance. Test scenario instances known as test cases are made up of a number of elements. An individual test case is stored based on its unique ID, the ID of the algorithm testers who created or edited the test case and whether the test case is stored locally or to the server. Other information stored include the name of the test case specified by the algorithm tester, an additional description of the test case and the date and time the test case was created and last modified The testInput tag stores the relevant test input information for instance whether the image-test-set is based on a tag, query or image-sequence. The final part of the XML document defines the language the imaging test is written in and the file address of where the test script is stored. The test script source is stored in the CDATA section within the XML document. All characters enclosed in the CDATA section are interpreted as characters, not markup or entity references. Where a test script is written in java, the filename and java class of the test script is specified there. The histogram test scenario instance serialised to an XML file is included in Appendix E.6.

5.3.2.2.2 Test Scenario Execution

Once a test scenario has been created by the algorithm tester, the application module called TestRuntime is responsible with running a test scenario inside the testing tool.

When a test scenario is executed by the TestRuntime, it receives a TestContext object which the test scenario can use to communicate with its testing environment, to access and use available imaging libraries and tools, in order to perform its required operations. For example the ImageIO library is used throughout the application to load, convert and save images. The TestRunTime Class source code is available in Appendix E.7.

5.3.2.2.2.1 Test Scenario Execution Sequence

After the relevant image test set is downloaded from the server if not stored locally, the test script is ran on each image in the image test set, the results of which are added to a test result set and returned to the test manager.

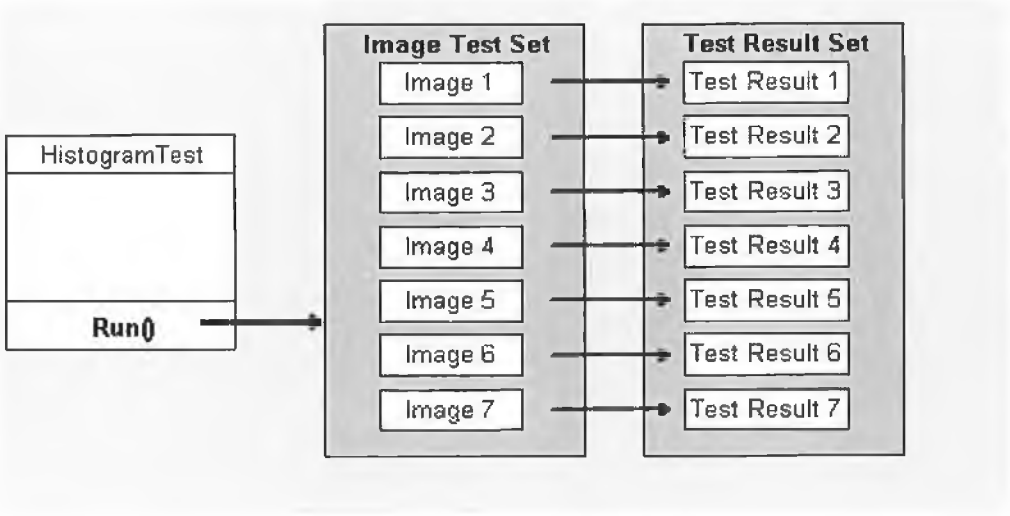


Figure 36: Algorithm executed on Image-test-set

5.3.2.3 Test Results

The TestRunHistory Class then provides functionality to build up a test result set and serialise it to an XML file for storage. The histogram test results XML file is included in Appendix E.8.

As will be examined in the next section the test result set can then be displayed in the testing tool's GUI to allow algorithm testers to assess the performance of the relevant algorithm characteristics.

5.4 ALGORITHM TEST FRAMEWORK GUI OVERVIEW

"*One way of making windows easier for users to control is to arrange for the relevant application program automatically to open the appropriate set of windows for each stage of the relevant tasks.*"(Preece et al, 1994:288)

The test framework is based on the Eclipse Workbench UI comprising of a separate perspective for each of the three tools.

- **Image Database Tool**: browse images, add new images, search for images, manage image queries, modify image properties, delete images
- **Testing Tool**: manage test scenarios, run test scenarios, save test results
- **Image Marker Tool**: mark images, edit marked images, unmark images

A perspective defines the set of editors and views for each tool arranged in an initial layout. The eclipse workbench UI then links the perspectives in an intuitive, easy-to-use window environment resulting in faster and more efficient testing practice.

"*This is a sensible solution for tasks for which the working set can be reliably specified in advance.*"(Preece et al, 1994:288)

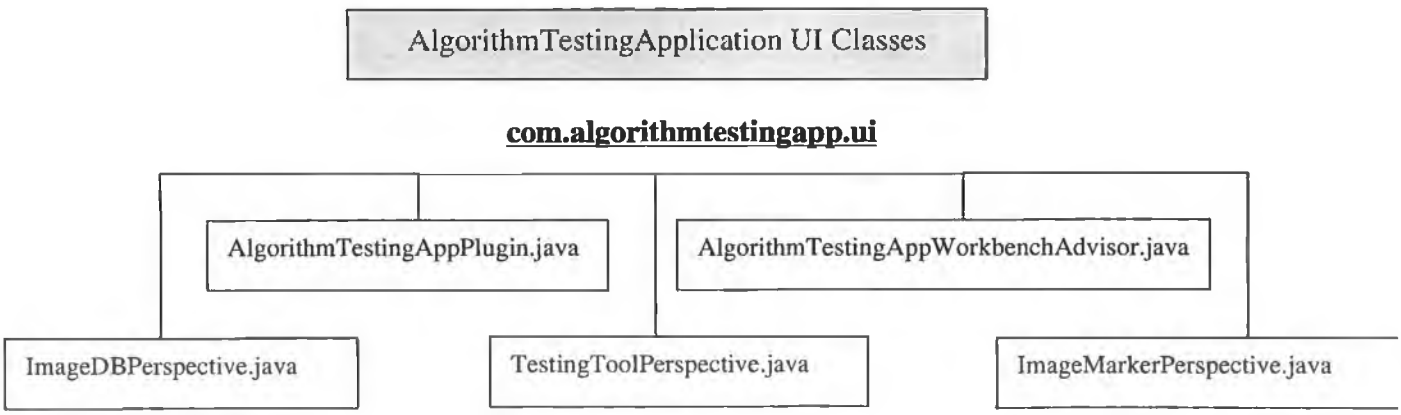


Figure 37: com.algorithmtestingapp.ui Package Structure

5.4.1 The Image Database Tool

From a conceptual point of view, the whole application revolves around images and the various test scenarios that will be performed on these images. The image database tool is used to store and manage images in the image database. It provides the algorithm tester with easy access to a comprehensive set of relevant test images that allow algorithms be tested on the most pertinent image-test-sets for a more complete evaluation (Jaynes et al, 2005:2). Figure 38 shows a screen shot of the image database tool.

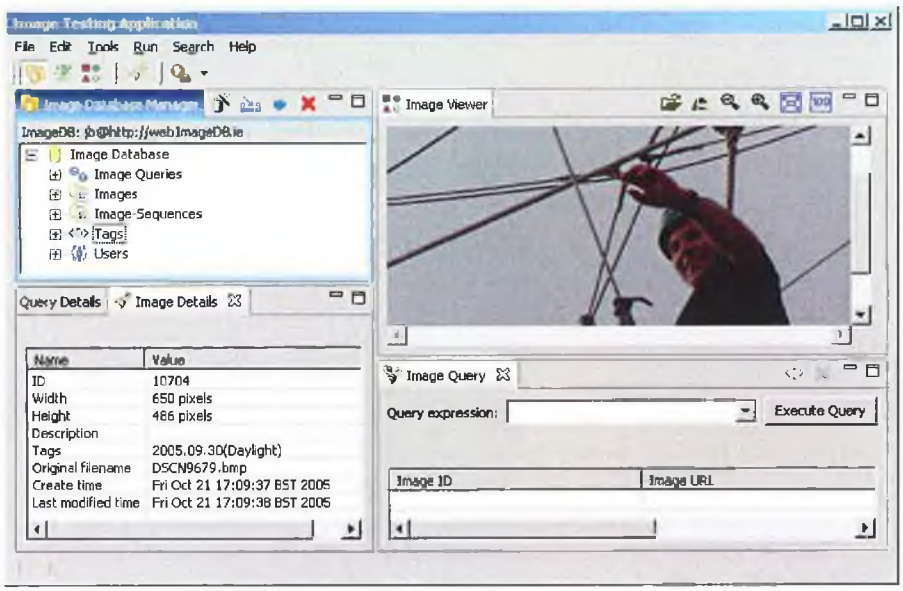


Figure 38: Image Database Tool

Within the database images may be divided into image test sets. An image set is a collection of images stored in the image database. To facilitate simple retrieval, an image set can be specified by either a query or a tag. Queries or tags allow algorithm testers to select a group of images from the database that have some common characteristic. Tags are associated or applied on images, and any image can have one or more tags associated with it. An Example of a tag: "RedEye_2MegaPixel_SonyCybershot". The database also contains items known as "queries", allowing the user to query the database and retrieve a set of images based on certain criteria. An example of a query: "Image.width >= 2000".

The testing tool retrieves image-test-sets from the database by specifying either a query or a tag.

5.4.2 The Marker tool

The job of a segmentation, classification and recognition algorithms is to detect certain features of interest within images. Therefore, for testing purposes, relevant features of interest should be marked on each image in the image-test-set prior to executing the algorithm. Accurate ground truth is crucial for effective performance evaluation of segmentation, classification and recognition algorithms (Black et al, 2002; Liu & Dov, 1999; Müller et al, 2004; Takeuchi et al, 2003:409). The image marker tool is used to generate ground truth by marking the relevant features of interest on the imported images as illustrated in Figure 39.

Figure 39: Image Marker Tool

The generated data or ground truth stored in Photographic Feature Markup Language (PFML) format is then used by the algorithm test execution process to analyse the performance of segmentation, classification and recognition algorithms.

5.5 GUI DESIGN FOR THE TESTING TOOL

5.5.1 THE TESTING TOOL OVERVIEW

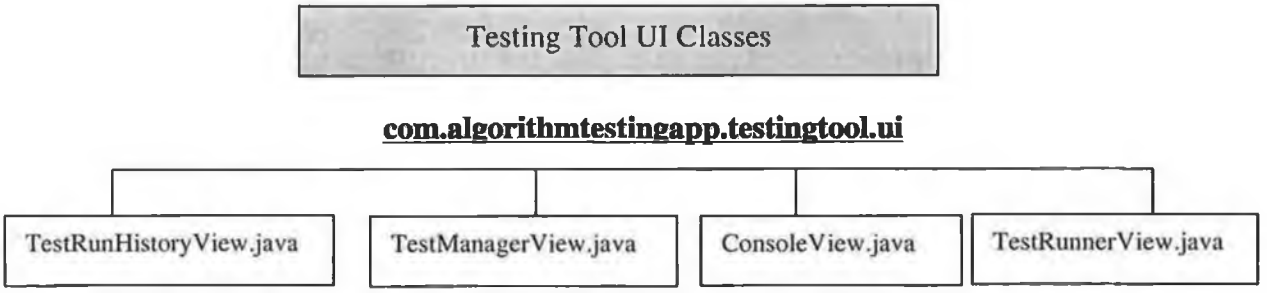


Figure 40: com.algorithmtestingapp.testingtool.ui package structure

Test scenario execution, the actual running of an algorithm on the image-test-set is the central point of this framework, around which all the other framework components are built. The testing tool allows the user to manage the creation and execution of test scenarios which produce the required algorithm performance information. The test framework like any Rich Client Platform inherits basic visual interfaces from Eclipse. The primary user interface building blocks of Eclipse rich client applications consist of view and editors (Shavor et al, 2003:315). Views within the Eclipse workbench are visual containers that allow users to navigate a hierarchy of information, open an editor, or display properties for the active editor.

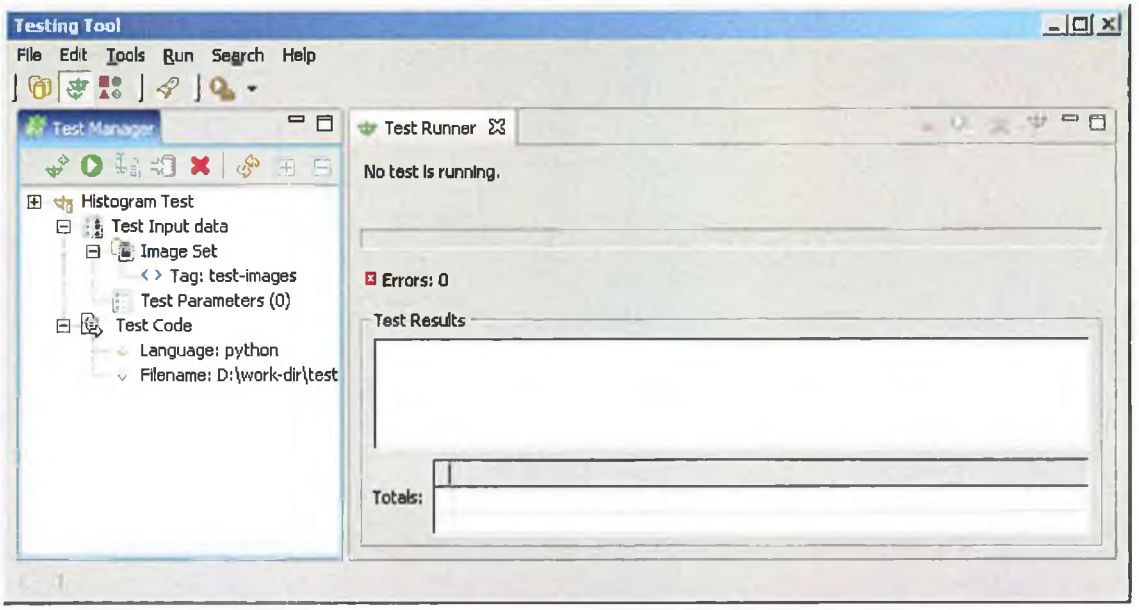


Figure 41: Testing Tool

As specified in the design chapter, the testing tool interface as shown in Figure 41 presents two main views, the "Test Manager" view and the "Test Runner" view. When the testing tool is first opened the "Test Manager" view on left hand-side automatically reloads and displays a tree hierarchy of existing test scenarios. A tree view is a selectable control that displays a hierarchal list of tree items that may be selected by the algorithm tester. In accordance with Raskin's principle (Raskin, 2000:105) for highlight indication and selection, the test scenario selected is highlighted, so the user knows at all times what the system thinks the user is pointing at. As displayed in Figure 41 additional test scenario information can be viewed by expanding the selected test scenario tree item. When a test scenario is ran information about the running status as well as the test results are displayed in the "Test Runner" view on right hand side.

5.5.2 Creating a new Test Scenario

A new test scenario is created by selecting the "Create New Test" context menu in the "Test Manager" view or by clicking on the "Create New Test" button in the "Test Manager"

view's toolbar. Since the test framework incorporates support for the integration of various image processing algorithms, on the first page of the wizard the algorithm tester must specify a name and description for the new test scenario and the code or test script that will be executed by the test scenario. The JFace toolkit offers a JFace wizard component, which coupled with other user-interface components within the Standard Widget Toolkit (SWT), provide a flexible mechanism to systematically gather user input and perform input validation. A wizard dialog is a specialized window for walking the end user through a sequence of steps; in this case the creation of a new test scenario. Figure 42 shows a screen shot of the "Create New Test" wizard.

"*The wizard itself acts as a controller, determining which wizard page, from a list of associated pages, is displayed in response to user interaction.*" (Shavor et al, 2003:301)

The dialogs have a standard layout: an area at the top, containing the wizard's title, description, and image; the actual wizard page appears in the middle and below it is a button bar containing Next, Back, Finish and Cancel buttons.

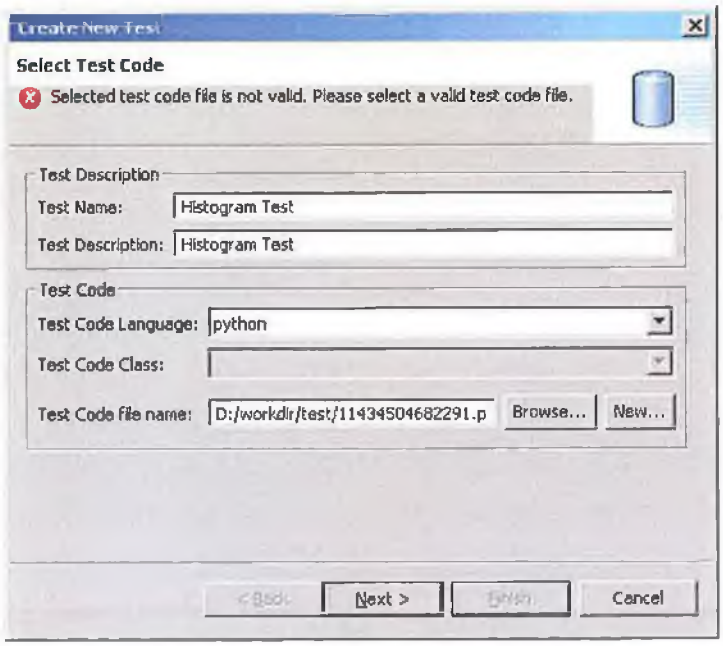


Figure 42: Create New Test Wizard

On the next page the input image test set for the test scenario must be selected and additional parameters for the test scenario may also be specified.

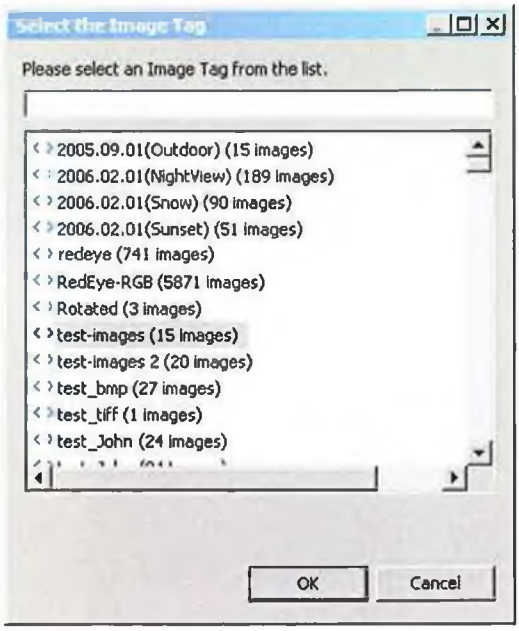


Figure 43: Image Test Set Input Wizard

An image set, a collection of images stored in the Image Database, can be specified by either a query or a tag. As displayed in Figure 43 queries or tags allow algorithm testers to select a group of images from the database that have some common characteristic. This allows for the natural selection of the most relevant image-test-sets for performance evaluation (Jaynes et al, 2005:2; Kong et al, 2005:111; Moon et al, 2002:7; Yang et al, 2002:52). Additionally it facilitates the decoupling of the algorithm from its operating conditions resulting in a more comprehensive algorithm assessment (Moon et al, 2002:1; Ojala et al, 2002:705; Phillips et al, 2000b).

5.5.3 Running a Test Scenario

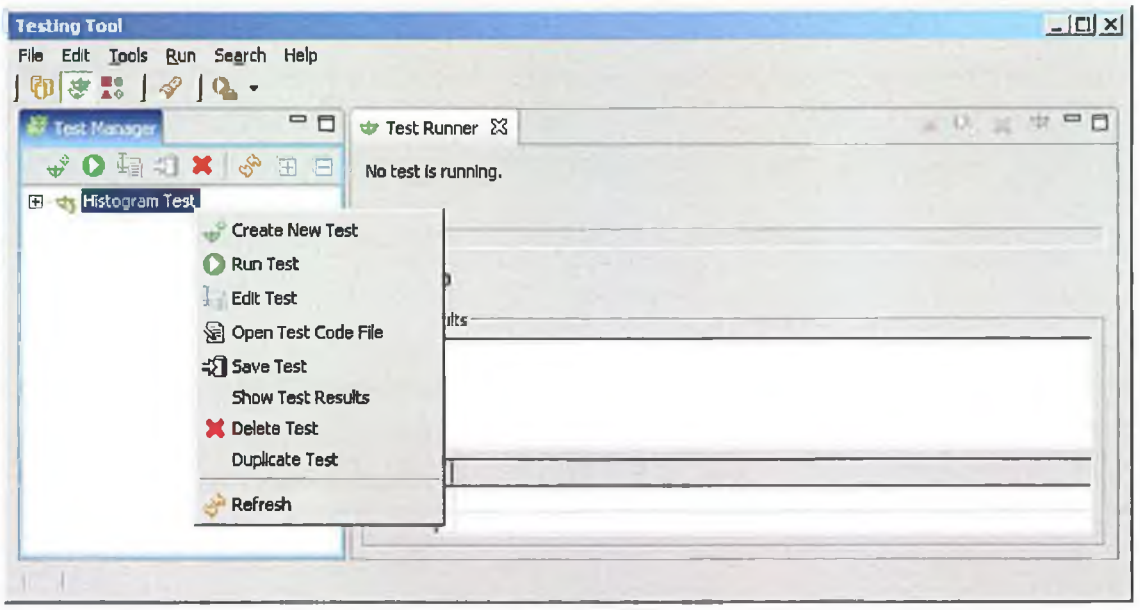


Figure 44: Running a test

Once a test scenario has been defined as required, it can be run to execute the required operations. As can be seen in Figure 44, the test scenario is selected in the "Test Manager" view, right clicked upon and "Run test" is clicked on in the context menu. Additional operations are available by right clicking on the test scenario and selecting an operation from the context menu. For instance as illustrated in Figure 45, test scripts written in any of the supported scripting languages can be edited and re-run instantly from within the testing tool by selecting "Open Test Code File". Providing scripting support allows both algorithm integrators and algorithm testers (non-programmers) write and edit scripts and helps to speed up the algorithm testing process.

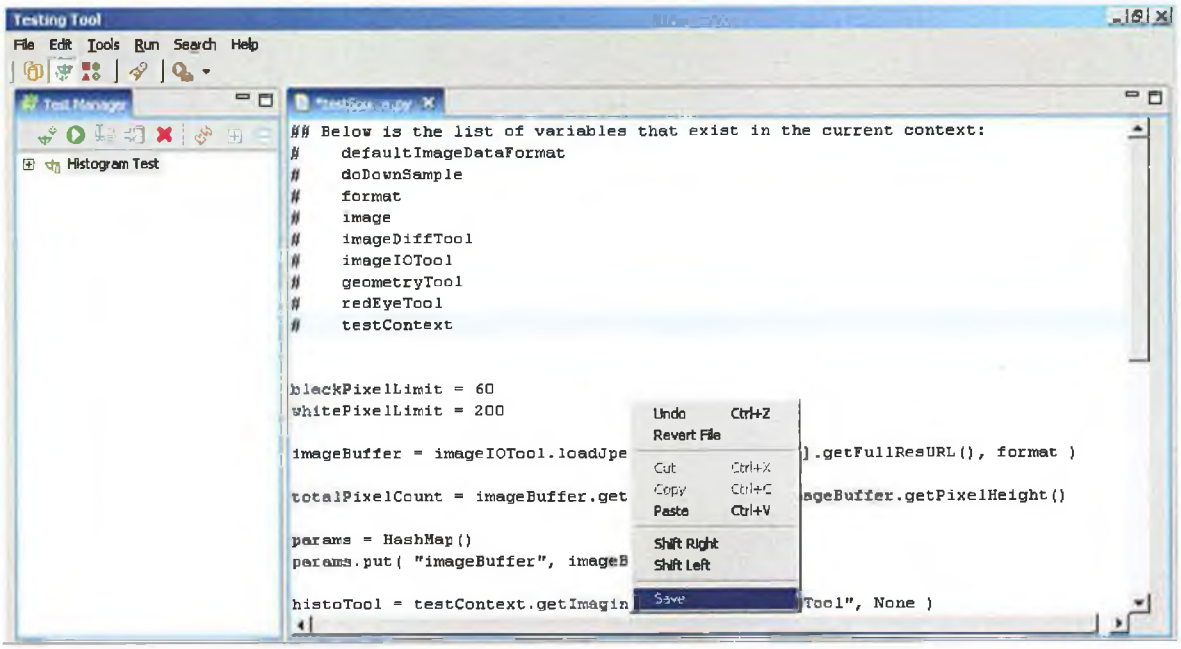


Figure 45: Editing a Test Script

5.5.4 View Test Results

A core part of the testing framework is the test-results – the data generated as a result of executing the test scenario. The "Test Runner" view displays information about the currently running test. Figure 46 displays a screenshot of a test scenario run in progress.

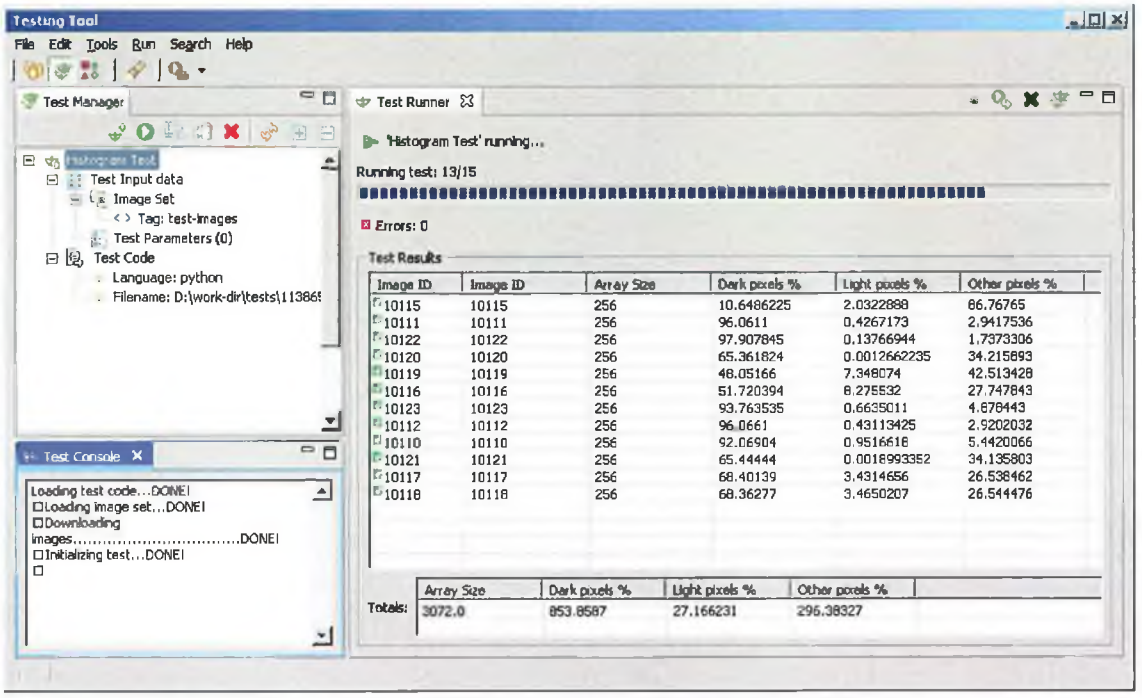


Figure 46: Test Runner View

To allow for investigation of algorithm performance on individual images as well as quantitative analysis of image-test-set performance, results for each image are displayed individually in rows depending on the criterion specified in the test script. A statistical measurement for the entire image test set is then presented in the totals table at bottom the "Test Runner" view. To help measure and compare algorithm performance, results are displayed in column format which contain the measurements and metrics specified in the test script.

"*The key design considerations are that the information displayed be legible and that it be easy for a user to locate and process*" (Preece et al, 1994:239)

(Raskin: 2000:76) notes the importance of response time and good feedback. In allowing for this, a progress bar is presented at top of test runner view to provide user with feedback on algorithm run. Additionally a "Test Console" view on bottom left of screen supplies relevant information on the running status of test scenario. In the "Test Runner" view,

execution of the currently running test scenario can be stopped at any time by selecting the "Stop current Test Run" button in the toolbar.

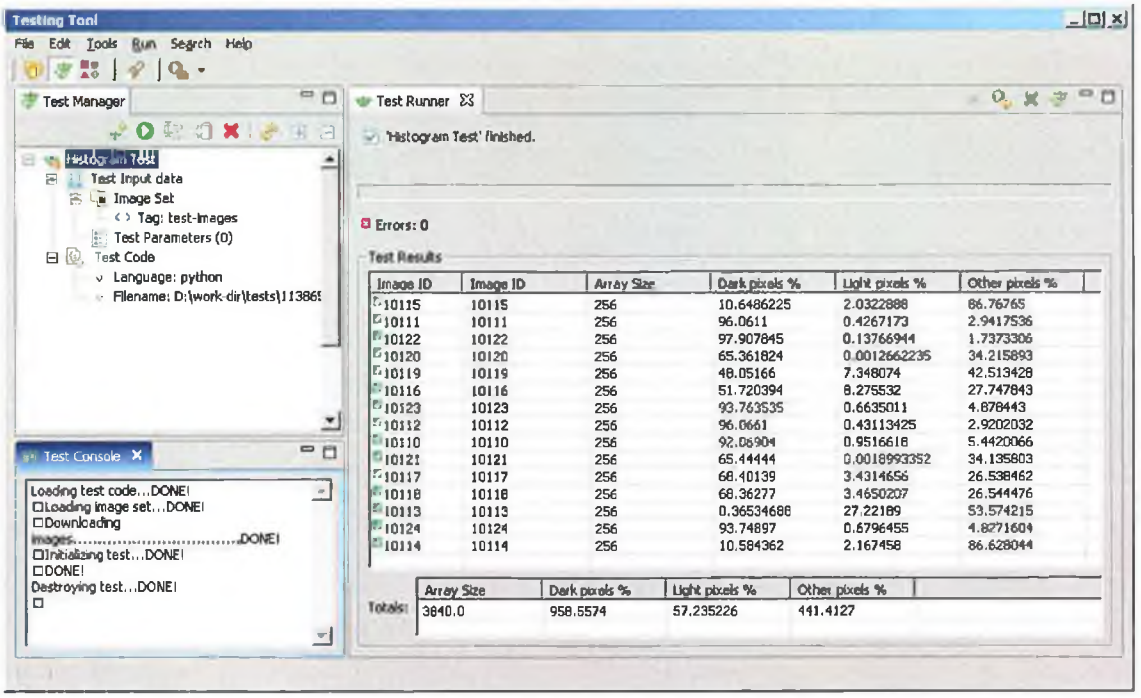


Figure 47: Test Scenario Execution Completed

After the test execution is successfully completed a simple table at the bottom of the "Test Runner" view displays the sum of the columns.

5.5.5 Test Results History

As mentioned in section 5.5.4, support is provided by the testing tool to store test-results. This allows for assessment of algorithm past performance by enabling the history of algorithm improvements iterations to be displayed. This means if the algorithm is changed to improve it, the test scenarios can be re-run, and the results from these can be compared with results from a previous version. The results will tell them reliably if the change in the algorithm is actually an improvement. In order to see the list of previous test run results for the currently loaded test scenario in the "Test Runner" view, the "Show results history"

button is pressed. A new tabbed view called "Test Run History" view, illustrated in Figure 48 is opened and the previous test run results for the current test scenario are displayed in it.

"*The tab approach can ease sorting of overlapping windows*" (Preece et al, 1994:293)

If previous test results become irrelevant they can be deleted by selecting them in the table in the "Test Run History" view and pushing the "Delete" button below the table.

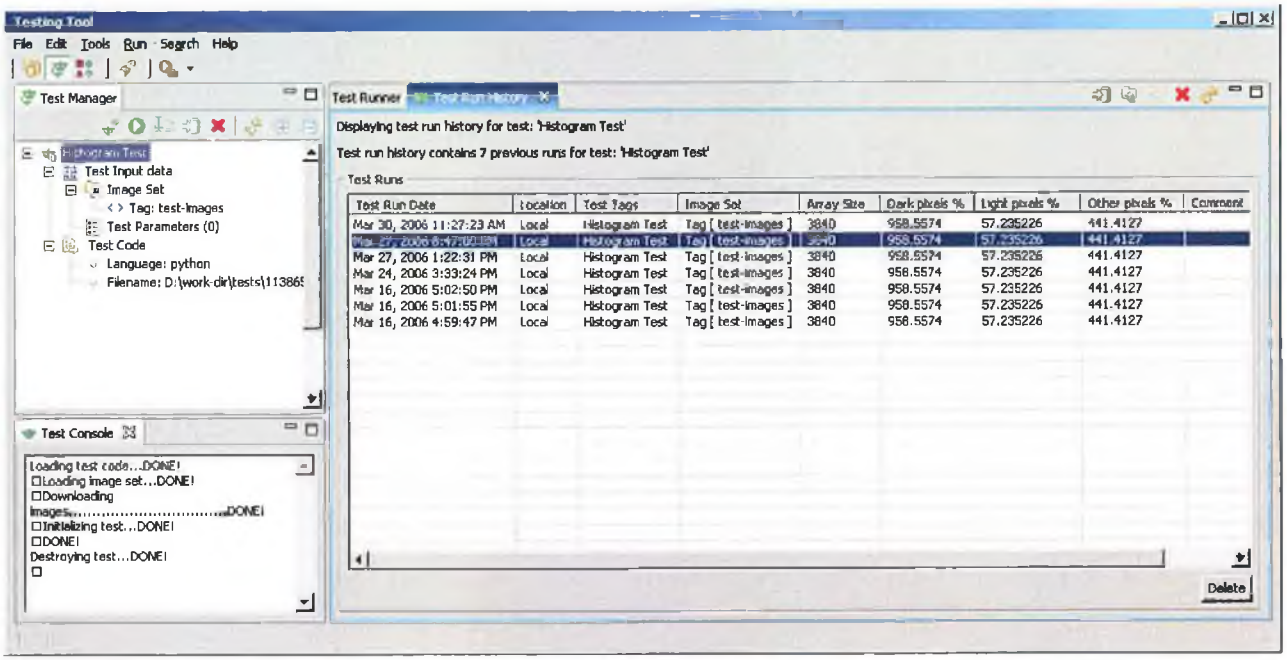


Figure 48: Test Results Display

5.5.6 Saving a Test on the Server

Additionally, newly created tests saved locally can be saved to the server by right-clicking on the test scenario, and then selecting the "Save Test" context menu item. The save operation can also be performed by clicking on the "Save selected Test in Server" button in the view's toolbar.

5.5.7 HELP

(Sommerville, 2001:340) reports that help systems are an important part of user interface design for the provision of user guidance.

"*Eclipse puts the Help system on equal footing by including it in its integration architecture.*" (Shavor et al, 2003:513)

Eclipse, the IDE used to develop the testing tool provides an integrated help environment. (Raskin, 2000:175) reports that a display of instructional text should be present the first time product is activated and by utilising Eclipse's help functionality a simple tutorial is provided on how to create, edit, save and run test scenarios which may be accessed at any time. In addition help documentation is provided in the form of HTML files which is integrated into the test framework architecture (Shavor et al, 2003:513).

5.6 SYSTEM IMPLEMENTATION EVALUATION

The system as implemented now represents a very efficient way to evaluate IP algorithm performance. The incorporated wrapper functionality provides a flexible mechanism to plug new algorithms in for testing purposes, allowing for the performance evaluation of a wide range of IP algorithms. The main advantages of the testing tool include:

- Making it easier to work with the complex technologies involved in testing image processing algorithms.
- It ties together discrete components of the algorithm testing process including image-test-sets, ground truth and performance metrics into the one integrated environment resulting in a faster and more efficient testing process.
- Algorithms can be easily evaluated by testers who may not have even developed the algorithms

And by following Raskin's (Raskin, 2000) principles for good interface design, the testing tool's UI should lead to higher productivity and increased customer satisfaction for the two

distinct categories of users identified, the algorithm integrator and algorithm tester. Additionally the testing tool user interface presented should lead to:

- Increased output
- Increased quality
- Decreased costs
- Decreased errors
- Decreased labour requirements
- Decreased production time

In both the IP algorithm testing and development process. The infrastructure can be deployed in modern algorithm development environment, where it will speed up testing cycles by quickly identifying problems early in an algorithm development lifecycle.

5.7 CHAPTER SUMMARY

"*Detailed and robust testing can provide insight into the underlying properties of algorithms*" (Zhao et al 2003:430)

This chapter has described some of the important implementation techniques employed during the development of the testing tool. With the core problem addressed by the testing framework being how to integrate various image processing algorithms for testing to support a semi-automatic testing processing the chapter started by describing the integration of an algorithm into the testing tool and the writing of test scripts incorporating the relevant metrics of interest to analyse individual algorithm performance. Following this, the development of the components which make up a test scenario and the underlying test execution architecture was explained. Finally the implementation of the overall test framework and the testing tool's Graphical User Interface was described.

CHAPTER 6: TESTING AND EVALUATION

Testing and Evaluation

6.1 INTRODUCTION

"*However, testing has become the preferred process by which software is shown, in some sense, to satisfy its requirements.*" (Hailpern & Santhanam, 2002:9)

Software Testing is the process used to identify the correctness, completeness and quality of a developed software application. According to (McGregor & Sykes, 2001), testing is important because it helps to ensure that the software application does everything it is supposed to do.

"*Testing is the primary method to ensure that programs comply with requirements.*" (Cheon et al, 2005:290)

The strategy for testing and evaluating the testing tool software solution adopted tactics recommended for object oriented systems. The two main methods used and described below, were usability testing and unit testing.

6.2 SOFTWARE TESTING

Although, during program development, software testing and coding were interleaved activities, (McGregor & Sykes, 2001) reports that development and testing processes are distinct, primarily because they have different goals and different measures of success. Software development aims to build a product that satisfies user needs whereas testing aims to answer questions about the product. Therefore testing and evaluation of the testing tool has been divided into a separate chapter here.

6.2.1 Software Inspections and Code Reviews

(McGregor & Sykes, 2001) reports software testing is typically accomplished by a combination of inspections, reviews, and test executions, the purpose of these activities being to observe failures. Therefore software inspections and code reviews were carried out during course of the SDLC and involved examining the source representation to uncover code defects and misunderstood requirements. Although inspections cannot check non-functional characteristics such as usability, it proved a very effective technique for discovering errors within the code.

6.2.2 Functional Testing

"*Software testing is the process of executing a software system to determine whether it matches its specification and executes in its intended environment.*" (Whittaker, 2000:77)

Software testing is the process of exercising a program with the intent to yield measurable errors (Cheon et al, 2005:290). During software development of the testing tool a particular kind of software testing, called unit testing was employed.

6.2.2.1 Unit Testing

Unit testing involves testing every program unit separately. In object-oriented programs, a unit can be a method, a class, or a set of closely associated classes. In the case of the testing tool, unit tests were performed on each of the main java classes as the classes were coded. The JUnit testing framework (www.junit.org, 2006) which is closely integrated with the Eclipse development environment was used to perform unit testing.

JUnit is a simple, open-source framework that allows users to produce and run unit tests. Firstly tests were defined by creating a subclass of the framework class junit.framework.TestCase and declaring methods in that class whose name starts with the letters "test". Then these tests were used to execute arbitrary code, and perform assertions on computed values by calling framework methods such as junit.assert.Assert.assertTrue(). A range of test assert methods are provided by JUnit. The

most generic method is assertTrue(), which simply passes or fails based on the value of a Boolean argument. A JUnit test implemented for the *HistogramWrapper* class *getHistogram* method is available in Appendix E.9.

6.3 USABILITY TESTING

"*Evaluation is concerned with gathering data about the usability of a design or product by a specified group of users for a particular activity within a specified environment or work context.*" (Preece et al, 1994:602)

Usability testing is a technique for ensuring that the intended users of a system can carry out the intended tasks efficiently, effectively and satisfactorily. According to (Preece et al, 1994:604) evaluations provide ways of answering questions about how well a design meets user needs.

"*Usability is tested to ensure that each category of user is supported by the interface; can learn and apply all required navigation syntax and semantics.*" (Pressman, 2005:569)

The purpose of carrying out usability testing on the testing tool was to determine what problems users might experience with the UI and to identify and eliminate faults. In addition the evaluation results helped to assess the efficiency of the testing tool interface and pinpoint any necessary modifications to improve its operation. Evaluations were carried out in two stages, formative evaluation and summative evaluation.

6.3.1 FORMATIVE EVALUATION

"*Formative evaluation provides information that contributes to the development of the system whereas summative evaluation is concerned with assessing the finished product.*"(Preece et al, 1994:613)

Due to the use of spiral development life cycle, formative evaluations took place throughout the testing tool design and development process. As mentioned in the design chapter (Section 4.9) initial ideas were prototyped using paper-based sketches and drawings. A low fidelity prototype is both inexpensive and easy to create, and provided valuable feedback at an early stage in the SDLC. By presenting low fidelity prototypes to stakeholders at the partner company and observing them as they were confronted with different tasks, areas of the testing tool interface that needed further improvement were

quickly identified. In addition flexible and semi-structured interviews were used to evaluate choices of initial design ideas and representations. Carrying out these evaluation functions early in the design process was vital to a smooth and stable development of the system.

6.3.2 SUMMATIVE EVALUATION

"*Interface evaluation is the process of assessing the usability of an interface and checking that it meets user requirements.*" (Sommerville, 2001:345)

In order to collect the opinions of the subjects on the interface and any issues with its design, summative evaluation was carried out as one of the final stages of the SDLC. A systematic evaluation of the testing tool's user interface design was deemed too expensive and an economically unrealistic process for this project, so the summative evaluation took the form of questionnaires administered to stakeholders at the partner company. Surveying users by using a questionnaire is a cheap but effective way of evaluating the software solution.

"*Questionnaires are used to actively study specific activities performed by users and get subjective information on the participants satisfaction.*" (Sommerville, 2001:345)

6.3.2.1 Usability Questionnaire

The usability questionnaire consisted of closed questions where respondent was asked to select an answer from a choice of alternative replies, and open questions, where the respondent was free to provide their own answer.

"*Questions in a questionnaire should be ambiguous and clearly laid out.*" (Preece et al, 1994:639)

The questionnaire was divided into five parts. Part one and two contained closed questions on the usability and functionality provided by the testing tool. A ratings scale was used to measure user satisfaction. Part three and four contained open questions relating to recommendations for changes to the testing tool UI and any additional comments regards the system. Finally part five contained closed questions on the overall system rating. After the questionnaire had been designed, it was administered to

stakeholders at the partner company who had been given the opportunity to use the testing tool beforehand. The main findings are presented below.

6.3.2.2 Usability Questionnaire: Main findings

6.3.2.2.1 Usability Ratings

The results from part one of the questionnaire demonstrated that the testing tool conformed to user requirements effectively and efficiently. Regards usability, users reported that they found the overall system easy to pick up and use with little instruction required. Users found the system allowed the testing of algorithms in a reasonably short time period. They found the selection of image-test-sets based on tags and queries reasonably straightforward and the "Test Manager" view made it easy to navigate through, select and execute tests. Information displayed in the "Test Runner" view was reported to be easy to read and understand and coupled with the information displayed in the Test Console view, provided enough detail to understand clearly and accurately, test progress information and algorithm performance results. Furthermore the display of test results in the "Test Run History" view made it easy to track the evolution of the algorithm from one version to another and compare algorithm performance between different versions.

6.3.2.2.2 Questions on Testing Tool Usability

Part two of questionnaire contained closed questions on specific functionality provided by the testing tool. Again a rating scale of one to five was used. Users found the testing tool UI to be structured in a logical and consistent manner and usable without continual help or instruction. Users also reported that the rules of interaction helped algorithm testers to work efficiently. Most importantly users found interaction with the testing tool to be simple.

6.3.2.2.3 Recommendations For Changes To The User Interface

Part three contained open questions relating to recommendations for changes to the testing tool interface. The main recommendations uncovered included:

"a. Pause/Resume test button. Reason: Some tests take a large amount of time and a lot of processing power. Sometimes you may need to pause the test so you can do other things at your computer.
b. There should be more things in post-processing of the tests. Like filter the images/sequences within some range of results (i.e.: FP > 20%) and throw them in a folder." (Algorithm Tester at Partner Company, Usability Questionnaire: March 06)

6.3.2.2.4 Any Additional Comments About The System

Part four of the questionnaire contained open questions on any additional comments from the stakeholders about the system. The main findings are summarised in the example answers provided by the stakeholders below.

"1. Does the new interface help to speed up the task of testing image processing algorithms?
Yes it does as the automatic tests replace the manually ones which take much longer.
2. Is it easy for the user to understand how to use the system based on it's visual appearance, or are some instructions required?
I believe it is easy." (Algorithm Tester at Partner Company, Usability Questionnaire: March 06)

These recommendations were then taken into consideration in determining future modifications to the testing tool.

6.3.2.2.5 Overall Ratings

Part five of the questionnaire elicited the overall ratings of system usability.

Efficiency	4
User friendliness	5
Pleasant to use	5
Easy to remember	4
Overall satisfaction	4
Potential future usage	5

Table 2: Overall Ratings from Usability Questionnaires

Table 2 summarises the overall results from the usability ratings. An example of one of the completed usability questionnaires is available in Appendix F.

6.4 CHAPTER SUMMARY

As can be seen in Table 2, feedback from the questionnaires proved very positive. The responses indicate the testing tool provides an efficient means of carrying out algorithm testing. The majority of users found the testing tool to be pleasant to use and a user friendliness rating of five indicates that users liked the UI and were not confused by using it. Overall satisfaction of users of the testing tool was high and a potential future usage rating of five points to the testing tool being used for algorithm testing well into the future.

CHAPTER 7: CONCLUSION AND FUTURE WORK

Conclusion and Future Work

7.1 CONCLUSION

The last 10 years have seen a digital image revolution, with soaring interest in image processing technology across the consumer and business landscape. The proliferation of digital cameras and mobile camera phones in today's world has resulted in a phenomenal surge in use of consumer digital images. New developments in image processing technology have not only affected individual lifestyle habits such as the way families interact through camera phones but has influenced all areas of science from tumor detection in biomedicine, to monitoring of weather patterns in environmental science and object and scene perception in robotic vision. Central to the adoption of digital photography by both businesses and consumers have been advances in the areas of image processing and image analysis, particularly in the development of complex image processing algorithms.

The rapid transition to digital photography has raised the expectations of non-professional photographers and for the digital imaging research community the ultimate goal is to allow consumers capture and manage discrepancy free images quickly and easily. Unfortunately, the digital image revolution has also brought some side effects. For instance the explosion in sales of camera phones means the problem of red eye, the most common customer complaint in the digital imaging market (Luo et al, 2004; Schettini et al, 2004:139; Smolka et al, 2003:1767; Zhang (b) et al 2004) has become more acute. Additionally because digital cameras perform differently than film cameras in their treatment of highlights and shadows, there has been an increase in the occurrences of

badly exposed photographs. And despite over 30 years of research in the area of computer vision, (Dalong et al, 2004:787) reports that robust recognition of faces in digital photographs, especially family photographs, still remains a challenge. Clearly although there have been major advancements in the field of image processing in the last 10 years, the performance of some of the most popular algorithms are still not satisfactory (Lu, 2003; Müller et al, 2004; Sharma & Reilly, 2003; Torres, 2004; Zhang et al, 2003).

With consumers yearning for sharper, brighter and clearer photographs and the security industry demanding perfect biometric technology the question arises, how do we test and compare algorithm performance. After conducting a survey of the literature on the methods and techniques being used, it can be seen that the field of image processing currently lacks a comprehensive testing framework, for assessing the performance of image processing algorithms. Proper evaluation has always been very important for any research area. Apart from the field of biometrics little emphasis has been put on algorithm performance evaluation up to now and where evaluation has taken place, it is been carried out in a somewhat cumbersome and unsystematic fashion, without any standardised approach. (Takeuchi et al, 2003:409) finds for instance that perhaps the only real performance evaluation measure in common use is longevity. The best algorithms being those that are accepted widely and implemented by many people for different applications.

7.2 SOFTWARE SOLUTION OVERVIEW

One key step towards the effective performance evaluation of IP algorithms is the provision an algorithm testing solution. Towards this end, this thesis has presented a comprehensive testing methodology and framework to adequately measure the performance of IP algorithms used in consumer digital imaging technology.

Inline with the principles set out in the FERET evaluation methodology (Phillips et al, 2000) the methodology exploits the direct connection between the algorithm being evaluated, the image-test-sets, and the actual testing protocol. An integrated database tool provides easy access to a comprehensive set of test images which allows algorithms to be tested on the most applicable image-test-sets resulting in a more complete algorithm evaluation (Jaynes et al, 2005:2; Kong et al, 2005:111; Moon et al, 2002:7; Yang et al, 2002:52). And because as (Chhabra & Phillips, 1998) suggest there is a lack of a systematic way of generating ground truth, the integrated image marker tool, allows for ground truth to be acquired quickly and accurately (Micheals & Boult, 2001:152; Müller et al, 2004).

The core problem addressed by the test framework is how to integrate different IP algorithms for testing to support a semi-automatic testing processing. As no framework can easily incorporate every IP algorithm, the testing tool provides an extensible interface so as to allow the analysis and performance assessment of a wide range of image processing algorithms. Essentially the job of integrating algorithms into the testing tool has been separating from the job of testing. This allows algorithm testers spend more time on understanding algorithm performance instead of spending time trying to implement algorithms for testing purposes.

Additionally the generic nature of the testing tool allows individual test scenarios to be specified based on algorithm and developers requirements resulting in quicker algorithm performance evaluation. A test scenario defines what parameters the algorithm takes as input, usually the image-test-set, what output the algorithm returns and how this is compared with ground truth to deduce performance scores (Föerstner, 1996:12). Within the testing tool, the display of test-results can be customised which facilitates fast and effective interpretation by both novice and expert users (Sharma & Reilly, 2003). The integrated nature of the overall test framework incorporating the database tool, the image

marker tool and testing tool in the one environment means that comprehensive algorithm performance evaluation is carried out quickly and easily. Comprehensive evaluations will produce more robust algorithms and in turn reduce algorithm development lifecycles.

The presented software solution has been in development for over two years and it is believed that the current release, addresses some of the major problems identified in algorithm testing practice. Initial user response has been very positive and indicates that the semi-automatic nature of the testing tool means that algorithm performance evaluation can now be carried out quickly and efficiently, thus greatly reducing testing time. Results from usability testing involving actual algorithm testers at the partner company have found that the new interface helps to speed up the task of testing image processing algorithms and in turn shorten the algorithm development lifecycle

Q: Does the new interface help to speed up the task of testing image processing algorithms?
A: Yes it does as the automatic tests replace the manually ones which take much longer."
(Algorithm Tester at Partner Company, Usability Questionnaire: March 06)

The technology can be deployed in modern algorithm development environment, where it will improve the standard of algorithm performance in terms of speed and accuracy and will also result in shorter algorithm development lifecycles.

7.3 FUTURE WORK

The results from usability tests carried out at the partner company suggest the testing tool provides a strong initial foundation for algorithm performance evaluation. Future work will continue to refine and enhance the application in response to user feedback. Since the overall test framework was developed in Eclipse, as the necessity arises, additional support tools related to testing IP algorithms can be easily integrated.

"*This approach is attractive as it allows applications to evolve over time by adding and replacing components.*" (McAffer & Lemieux, 2005)

The main aim of the testing tool is to produce relevant algorithm performance data.

(Sharma & Reilly, 2003) suggests any performance evaluation solution should present data graphically to facilitate fast and effective interpretation. Mechanisms for displaying more information in the "Test Runner" and "Test Run History" views of the testing tool may be added in the future. Another related improvement would be to provide a better Connection between the testing tool and the image marker tool. For instance, if the algorithm tester clicked on false positive in the "Test Runner" view of the testing tool, the marking tool should open to display the incorrectly detected feature in the offending image. This would mean that algorithm testers would be able to pinpoint algorithm performance problems quicker and more effectively. Additionally, to effectively analyse complex algorithm performance results such as those produced from more technical algorithms, more sophisticated reporting tools should be incorporated into the testing tool. A reporting capability would mean that detailed reports could be produced and shared between algorithm development and QA teams to help streamline and speed up the testing process.

The core problem addressed by the testing tool is how to integrate different image processing algorithms for testing to support a semi-automatic testing process. The testing tool provides an extensible interface so as to allow the analysis and performance assessment of a wide range of image processing algorithms. It would be worth researching the possibility of automating the process of integrating algorithms into the testing tool. Making it easier to write algorithm wrappers would certainly speed up the testing process.

APPENDIX A: SPIRAL LIFE CYCLE MODEL

The Spiral life cycle model addresses the shortcomings of the Waterfall model by presenting an incremental development process, in which developers repeatedly assess changing project risks to manage unstable requirements (Nuseibeh, 2001:115). The model created by (Boehm, 1988) has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through each of these phases in iterations called Spirals in the model. With each iteration around the spiral, beginning from the centre and working outwards, progressively more complete versions of the system are developed. (Green & DiCaterino, 1998:8).

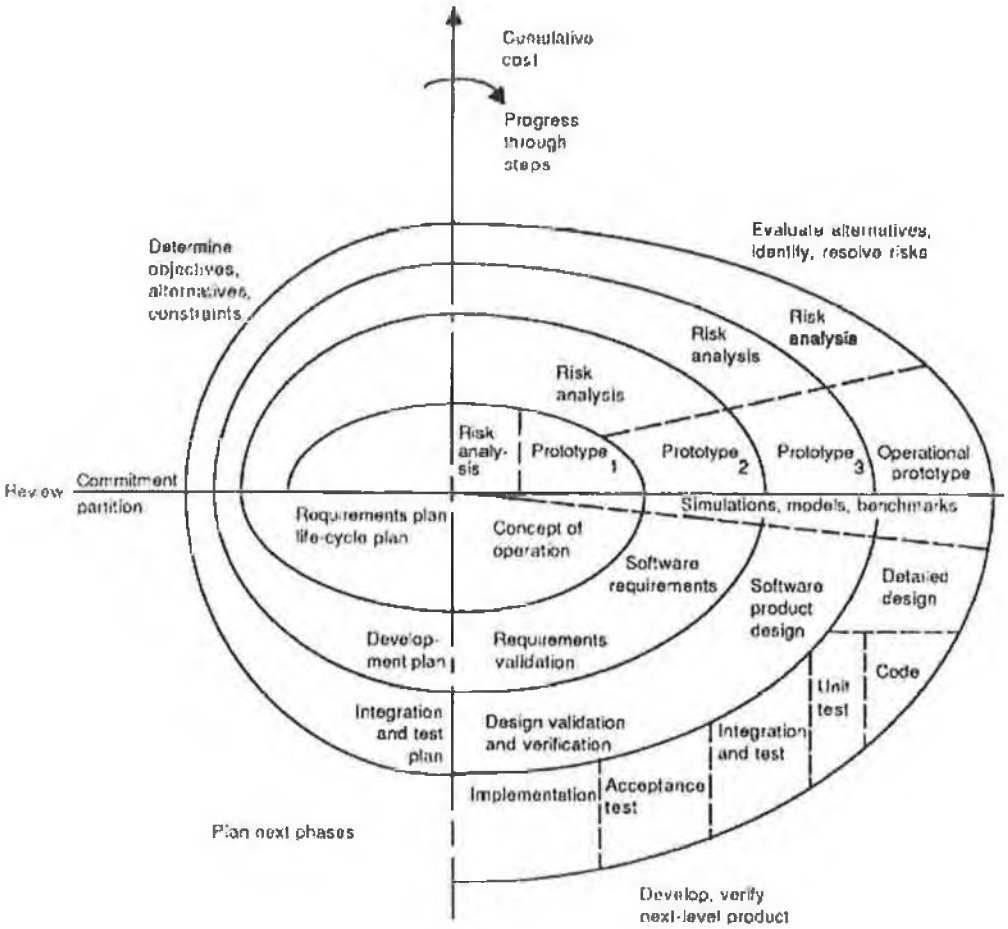


Figure 49: Spiral Model of the Software process (Source: Boehm, 1988:64)

APPENDIX B: ECLIPSE RICH CLIENT PLATFORM

"*It has become one of the most flexible, powerful, and integrated Java development environments.*" (Yang et al, 2005:1)

Although Eclipse was originally built as an integrated development environment for software development, the Eclipse platform can also be used to build client applications (Carlson, 2005; Gruber et al, 2005:289; McAffer & Lemieux, 2005). Fundamentally, Eclipse is a framework for plug-ins (Yang et al, 2005:2). A plug-in being the foundation building block that is used to add new functionality to the environment. The Eclipse Rich Client Platform (RCP) is a subset of Eclipse that allows a set of plug-ins to be developed and deployed as a standalone application, independent of the Eclipse development environment. The minimal set of plug-ins required to develop a rich client application is collectively known as the Rich Client Platform (RCP). Essentially, the RCP offers a generic Eclipse workbench that developers extend to construct client applications.

"*The RCP is a natural progression toward integrating not only tools but also applications and services.*" (Gruber et al, 2005:289)

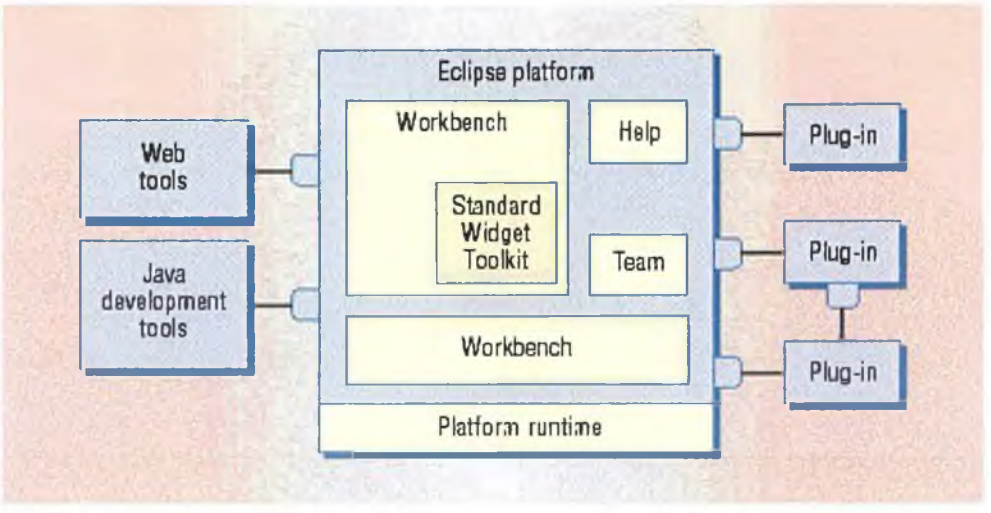


Figure 50: Eclipse Rich Client Platform - Source: Geer, 2005:17

As illustrated in Figure 50 the Rich Client Platform is composed of the following components:

- The Eclipse runtime which provides the foundational support for plug-ins.
- The workbench is the "UI personality" of the eclipse platform which incorporates the editors, views, and perspectives that will make up the environment that is the overall test framework (Shavor et al, 2003:198).
- The Standard Widget Toolkit (SWT) and JFace toolkit for building user interfaces.

APPENDIX C: EXTENSIBLE MARKUP LANGUAGE

XML is a simple, very flexible format derived from Standard Generalised Markup Language (SGML). It is a standardised meta-language designed to store, carry and exchange data and can be used to:

- Define data structures
- Define tags
- Make these structures platform independent
- Process XML defined data automatically

"*XML simply defines standard ways to manage and exchange complex documents.*" (Orfali et al 1999:625)

XML is completely flexible in how its data can be structured so can be used describe any kind of information (Hunter et al, 2000: 21). (Knudsen & Niemeyer, 2005) reports that XML does for content what Java did for programming by providing a portable language for describing data. It is designed specifically for storing and transmitting data from one place to another (Hunter et al, 2000:1, Maruyama et al, 1999, Young, 2000:3) and (Vaughan-Nichols, 2003b:14) reports XML has become an integral part of numerous important technologies for exchanging information between systems. Since XML data is stored in plain text format, it can be easily moved between platforms (McLaughlin, 2001:2). XML Path language - XPath a querying language can be used for searching for a particular piece of information in an XML document (Hunter et al, 2000: 21).

APPENDIX D: RED EYE DETECTION ALGORITHM OVERVIEW

Detection and recognition algorithms can make two types of errors; false positive in which regions are not classified correctly, resulting in low detection rates and false positives in which regions are mistakenly detected (Liu & Dori, 1999:103; Martin et al, 1997; Yang et al, 2002:35). Accordingly it must be appreciated that there is always a trade-off between true positive and false positive detection (Clark & Clark, 2002:4; Liu & Dori, 1999:103; Martin et al, 1997). If detection rules are to detailed the algorithm may fail to detect regions that do not pass rules whereas if rules are too general algorithm may return false positives. For example currently the main challenge in effective red eye correction is the avoidance of false positives, misclassifying red spots found on the image as red eyes, while maintaining high correction rates and quality (Schettini et al, 2004). With metrics of interest defined by the algorithm integrator to obtain and track these values, algorithm testers can then train and tune their algorithm to get the balance right in algorithm development(Courtney & Thacker, 2001:5; Yang et al, 2002:35).

Therefore to test a red eye detection algorithm, for each image, the location of every red eye within is identified or marked prior to running the red-eye detection algorithm. Images have to be marked in order to automatically conclude on the efficiency of the algorithm. The red eye detection algorithm proves correct if it identifies red-eyes in the same location as the marked areas on each image. The comparison between the original marked images and the images processed by an algorithm tells if that algorithm works correctly (Liu & Dori, 1999:98).

Stage 1 – Red Eye Detection Test Scenario

1. Run the red eye detection algorithm on an image – a list of the detected red eyes - features of interest are returned (List of Rectangle Co-ordinates)

2. Compare these (result of step 1) with the list of manually marked red eyes prepared by the image Marker Tool (List of Rectangle Co-ordinates)

Stage 2 – Red Eye Detection Test Scenario

This will then give:

a. a list of matching Red eye (A marked red eye over-laps well enough with a detected red eye)
b. list of marked red eyes that have not been detected
c. list of detected red eyes that do not match any marked red eyes

Stage 3 – Red Eye Detection Test Scenario

Results in 4 Columns:

1. No. of marked red eyes
2. No. of correctly detected red eyes (the value from a.)
3. False negatives (the value from b.)
4. False positives (the value from c.)

APPENDIX E: SOURCE CODE

APPENDIX E.1: IMAGINGTOOL INTERFACE

```
/**
 * This interface defines an imaging tool, and it must be implemented
 * by all tools that provide imaging services.
 */
public interface ImagingTool {

   /** Returns the name of this tool. */
   public String getName();

   /** Returns the version of this tool. */
   public String getVersion();

   /**
   * Initialises the imaging tool.
   *
   * @param bundleManager the library bundle manager that can be used
   * to access various bundles.
   * @return true is the initialisation succeeded, false otherwise.
   */
   public boolean init( LibraryBundleManager bundleManager );

   /**
   * Executes the image processing operation with the specified name
   * and with the specified input parameters; returns the results of
   * the operation; each imaging tool can return its own result type.
   *
   * @param operationName the name of the operation to perform.
   * @param inputParams the input parameters.
   * @return returns the results of the operation, each imaging tool
   * can return its own result type.
   *
   * @throws ExecuteOperationException if executing the operation
   * fails for some reason.
   */
   public Object executeOperation( String operationNme, Map InputParam)
            throws ExecuteOperationException;
   /**
   * Destroys the imaging tool.
   */
   public void destroy();
}
```

APPENDIX E.2: HISTOGRAM.H

```
#ifndef __HISTO_H__
#define __HISTO_H__

#include "ImageIO.h"

#ifdef __cplusplus
extern "C" {
#endif

#if !defined(HISTCALL)
# if defined(_STDCALL_SUPPORTED)||defined(_MSC_VER)&&(_MSC_VER >= 800)
#  define HISTCALL __stdcall
# else
#  define HISTCALL
# endif
#endif

// Type to return the status of execution of a function.
typedef INT32 HIST_STATUS;

#define HS_OK 0
#define HS_FAIL -1 //Unspecified error.
#define HS_NO_SYSTEM_RESOURCES -2 // not enough system resources.
#define HS_INVALID_ARG -3 // invalid argument

// Defines
#define IN
#define OUT
#define OPTIONAL

/**
 * Calculates and returns the histogram for the specified crop from the
 * specified image.
 *
 * @param image the target image
 * @param x the horizontal coordinate of the top-left point of the crop
 * rectangle.
 * @param y the vertical coordinate of the top-left point of the crop
 * rectangle.
 * @param width the width of the crop rectangle
 * @param height the height of the crop rectangle
 * @param result an int [256] array that contains the result
 */
HIST_STATUS HISTCALL GetHistogram( IN IO_Img* image,
      IN UINT32 x,
      IN UINT32 y,
      IN UINT32 width,
      IN UINT32 height,
      OUT UINT32* result
```

```
      );

#ifdef __cplusplus
}
#endif
#endif // __HISTO_H__
```

APPENDIX E.3: HISTOGRAMWRAPPER.H

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class HistogramWrapper */

#ifndef _Included_HistogramWrapper
#define _Included_HistogramWrapper

#ifdef __cplusplus
extern "C" {
#endif

/*
* Class: HistogramWrapper
* Method: getHistogram
* Signature: (Lcom/imagetestingapp/tools/imageio/ImageBuffer;IIII)[I
*/
JNIEXPORT jintArray JNICALL Java_HistogramWrapper_getHistogram
(JNIEnv *, jobject, jobject, jint, jint, jint, jint);
#ifdef __cplusplus
}
#endif
#endif
```

APPENDIX E.4: HISTOGRAMWRAPPER.CPP

```
#include <jni.h>
#include "HistogramWrapper.h"
#include "ImageIO.h"
#include "histogram.h"

// Utility function used to throw and exception by name.
static void throwByName( JNIEnv *env,const char *name,const char *msg){
   jclass cls = (env)->FindClass( name );

   // if cls is NULL, an exception has already been thrown
   if( cls != NULL ) {
      (env)->ThrowNew( cls, msg );
   }
```

```
    (env)->DeleteLocalRef( cls ); // free the local ref
}

// Utility function used to throw an exception based on a status code.
static void throwByStatus( JNIEnv *env, HIST_STATUS status ) {
    switch( status ) {
        case HS_FAIL:
            throwByName( env, "java/lang/RuntimeException","DLL returned
                  Unspecified error." );
            break;
        case HS_NO_SYSTEM_RESOURCES:
            throwByName( env, "java/lang/OutOfMemoryError",
                  "DLL returned There's not enough system resources." );
            break;
        case HS_INVALID_ARG:
            throwByName( env, "java/lang/IllegalArgumentException",
                  "DLL returned Bad argument." );
            break;
    }
}

/**
* Class: HistogramWrapper
* Method: getHistogram
* Description:
*/
JNIEXPORT jintArray JNICALL
Java_com_imagetestingapp_tools_HistogramWrapper_getHistogram
    ( JNIEnv * env, jobject obj, jobject imageBuf, jint x, jint y, jint
    width, jint height )
{
    // variable declarations
    jclass cls;
    jmethodID mid;
    jlong imageStructPointer;
    IO_Img* pIOImg = NULL;
    jintArray histogram = NULL;
    UINT32 data[ 256 ];
    if( imageBuf == NULL ) {
        throwByName(env,"java/lang/NullPointerException","imageBuf
        argument is null" );
        return NULL;
    }
    // Get the class of the ImageBuffer object needed to access its
          members.
    cls = (env)->GetObjectClass( imageBuf );
    if( cls == NULL ) {
        return NULL; /* class not found, exception already thrown */
    }

    // Get ID of the "getImageStructPointer" method of the ImageBuffer
```

```
    mid = (env)->GetMethodID( cls, "getImageStructPointer", "()J");
    if( mid == NULL ) {
       return NULL; /* exception already thrown */
    }

    // Call the "getImageStructPointer" method of the ImageBuffer object
    imageStructPointer = (env)->CallLongMethod( imageBuf, mid);
    if( imageStructPointer == NULL ) {
       throwByName( env, "java/lang/IllegalArgumentException",
                    "imageStructPointer is NULL" );
       return NULL;
    }
    pIOImg = (IO_Img*)imageStructPointer;

    HIST_STATUS status = GetHistogram( pIOImg, x, y,width,height,data );
    if( status == HS_OK ) {
       histogram = (env)->NewIntArray( 256 );
       if( histogram == NULL ) {
          return NULL; /* out of memory error already thrown */
       }
       for( int i = 0; i < 256; i++ ) {
          jint val = data[ i ];
          (env)->SetIntArrayRegion( histogram, i, 1, &val );
       }
    }
    else {
       throwByStatus( env, status );
    }
 return histogram;
}
```

APPENDIX E.5: TESTMANAGER CLASS

```
public class TestManager {

    private static TestManager singleInstance;

    static {
        singleInstance = new TestManager();
    }
    /**
*  Constant  that  contains  the  name  of  the  folder  where  the  tests
* will be stored in the work directory.
     */
    public static final String TEST_STORE_DIR_NAME = "tests";

    private File testStoreDir;

    /**
     * This is a flag that specifies if the TestManager instance was
```

```
     * initialised.
     */
    private boolean initialised = false;

    /**
     * Holds existing tests, maps test ids to TestDefinition instances.
     */
    private Map testsMap;

    /**
     * Creates a TestManager instance.
     */
    private TestManager() {
        this.testsMap = new HashMap();
    }

    /**
     * Returns the single instance of this class.
     *
     * @return the single instance of this class.
     */
    public static TestManager getInstance() {
        return singleInstance;
    }
```

APPENDIX E.6: HISTOGRAM TEST SCENARIO XML INSTANCE

```
<?xml version="1.0" encoding="UTF-8" ?>
- <testDefinition type="testCase" id="14016" clientId="11386584335491" storedOnServer="true">
    <serverLastModified>1138662658313</serverLastModified>
    <name>Histogram Test</name>
    <description>Histogram Test</description>
    <created>1138658433518</created>
    <lastModified>1138662658313</lastModified>
  - <testInput>
    - <imageSet type="tag">
        <tag name="test-images" type="Image" expression="//tag[@name='test-images']" />
      </imageSet>
    </testInput>
  - <testCode language="python">
      <filename>D:\ImageTestingApplication\work-dir\tests\testSource.py</filename>
    - <sourceCode>
      + <![CDATA[  ]]>
      </sourceCode>
    </testCode>
  </testDefinition>
```

APPENDIX E.7: TESTRUNTIME CLASS

```
/**
* This class is responsible with running an imaging test.
*/
public class TestRuntime {

/**
* Creates a TestRuntime instance with the specified configuration.
*
* @param runtimeConfig the test runtime configuration.
* @throws TestRuntimeException if the test runtime initialisation
fails.
*/
public TestRuntime( TestRuntimeConfiguration runtimeConfig )
throws TestRuntimeException;

/**
* Runs the specified test. A request to cancel the test run should be
* honored and acknowledged by throwing
*<code>InterruptedException</code>.
*
* @param testDef the test definition.
* @param listener the listener for this test run.
* @return the result of the test run.
```

```
*
* @throws TestRunException if some error occurs while running the test.
* @throws InterruptedException if the test runtime detects a request to
* cancel, using <code>TestRunListener.isCanceled()</code>, it should
* exit by throwing InterruptedException.
*/
public TestResultSet runTest( TestDefinition testDef,
TestRunListener listener )
throws TestRunException, InterruptedException;

}
```

APPENDIX E.8: HISTOGRAM TEST RESULT XML INSTANCE

```
<?xml version="1.0" encoding="UTF-8" ?>
- <testResultSet testId="14016" testRunTimestamp="1143488820483" storedOnServer="false">
  - <summary>
    - <tags>
        <tag>Histogram Test</tag>
      </tags>
    - <testInput>
      - <imageSet type="tag">
          <tag name="test-images" type="image" expression="//tag[@name='test-images']" />
        </imageSet>
      </testInput>
    - <resultTotals>
      - <testResult type="success">
          <testResultField name="Image ID" value="" type="string" />
          <testResultField name="Array Size" value="3840" type="int" />
          <testResultField name="Dark pixels %" value="958.5574" type="float" />
          <testResultField name="Light pixels %" value="57.235226" type="float" />
          <testResultField name="Other pixels %" value="441.4127" type="float" />
        </testResult>
      </resultTotals>
    </summary>
  - <testResult inputDataType="image" inputDataId="10115" type="success">
      <testResultField name="Image ID" value="10115" type="string" />
      <testResultField name="Array Size" value="256" type="int" />
      <testResultField name="Dark pixels %" value="10.6486225" type="float" />
      <testResultField name="Light pixels %" value="2.0322888" type="float" />
      <testResultField name="Other pixels %" value="86.76765" type="float" />
    </testResult>
```

APPENDIX E.9: GETHISTOGRAM METHOD JUNIT TEST

```
/**
 * Test case for the getHistogram method.
 */
public void testGetHistogram() {
  LibraryBundle bundle = this.dllBundleManager.getBundle( "FNhisto",
                LibraryBundleManager.LATEST_VERSION );

  if( bundle == null ) {
    fail( "The required FNhisto library is not available" );
    return;
  }

  bundle = this.dllBundleManager.getBundle( "HistogramWrapper",
                LibraryBundleManager.LATEST_VERSION );

  if( bundle == null ) {
    fail( "The required HistogramWrapper library is not available" );
    return;
  }

  ImagingTool imagingTool = new HistogramWrapper();
  if( ! imagingTool.init( dllBundleManager ) ) {
    fail( "Failed to initialise the HistogramWrapper" );
  }

  int[] histogram;
  Rectangle cropRect = new Rectangle();

  try {
    Map params = new HashMap();
    imagingTool.executeOperation( "getHistogram", params );
    assertTrue( "Test Failed, executeOperation - getHistogram should
have thrown a NullPointerException", false );
  }catch( NullPointerException e ) {
    // Success, we should get a NullPointerException
  }catch( ExecuteOperationException e ) {
    assertTrue( "Failed with exception: " + e, false );
  }

  ImageBuffer imageBuffer = loadRBGImageBuffer(  this.testImageFile );

  try {
    Map params = new HashMap();
    params.put( "imageBuffer", imageBuffer );
    params.put("cropRect",new Rectangle( 0, 0,
imageBuffer.getPixelWidth(), imageBuffer.getPixelHeight() ) );
```

```
    Object result =  imagingTool.executeOperation( "getHistogram",
params );
    assertTrue( "getHistogram did not return  int[]", result instanceof
int[] );
    histogram = (int[]) result;
    assertNotNull( "getHistogram returned a null int[]", histogram );
    assertTrue( "getHistogram returned an int[] with length != 256",
histogram.length == 256 );
    for( int i = 0; i < histogram.length; i++ ) {
      System.out.println( "histogram[" + i + "] = " + histogram[ i ] );
    }
    }catch( ExecuteOperationException e ) {
      assertTrue( "Failed with exception: " + e, false );
    } finally {
      imageBuffer.free();
      imageBuffer = null;
      Runtime.getRuntime().gc();
    }
 }
```

APPENDIX F: COMPLETED USABILITY QUESTIONNAIRE

Usability Questionnaire – "The Testing Tool"

1. Usability Ratings

Please provide ratings for following functionality:

Rating scale to be used:

1 - Strongly disagree

2 - Disagree

3 - Not sure

4 - Agree

5 - Strongly agree

I found it easy to pick up and use the system with little instruction required.

5

The job of running algorithm tests was intuitive and easy to use

4

The system responded quickly and allowed me to test algorithms in a reasonably short period

4

The steps involved in creating a new test, the selection of the appropriate image-test-set and defining of appropriate test parameters were reasonably straightforward.

1

The selection of image-test-sets based on tags and queries was reasonably straightforward.

4

The steps involved in the selection of a test and actual test execution were reasonably straightforward.

3

The steps involved in editing an existing test were reasonably straightforward

3

The steps involved in deleting an existing test were reasonably straightforward

5

The Test Manager view was easy to read and understand

5

The Test Manager view made it easy to navigate through tests

5

It was always reasonably easy to tell which test was currently selected.

5

The Test Runner view which displays test run progress information and test results, provided enough detail for me to understand clearly and accurately, test progress information and test performance results.

4

In the Test Runner view information displayed in the test run progress area was easy to read and understand.

3

The information displayed in the Test Console view at bottom left of screen was useful, easy to read and understand.

3

The information displayed in the Test Run History view was easy to read and understand.

5

The display of test results made it easy to track the evolution of the algorithm from one version to another and see compare algorithm performance between different versions.

5

It was easy to save the selected test to the server.

5

It was easy to save the selected test results to the server.

5

I would find such the system useful in my day to day job when I need to test image processing algorithms on a large quantity of images.

4

The testing tool greatly speeds up the algorithm testing process.

4

2. Questions on Testing Tool Usability.

Is the testing tool usable without continual help or instruction?

5

Do the rules of interaction help a knowledgeable user to work efficiently?

3

Do interaction mechanisms become more flexible as users become more knowledgeable?

3

Has the testing tool been tuned to the physical and social environment in which it will be used?

3

Is the user aware of the state of the testing tool? Does the user know where he/she is at all times?

3

Is the testing tool interface structured in a logical and consistent manner?

5

Are interaction mechanisms, icons, and procedures, consistent across the testing tool interface?

5

Does the interaction anticipate errors and help the user correct them?

2

Is the testing tool interface tolerant of errors that are made?

3

Is the interaction simple?

5

3. Recommendations for changes to the Interface

What buttons, if any, did you find least useful? Most useful?

Lest useful: expand all

Most useful: start test

Are there any buttons (functions) that you would add to the system? Why?

a. Pause/Resume test button. Reason: Some tests take a large amount of time and a lot of processing power. Sometimes you may need to pause the test so you can do other things at your computer.

b. There should be more things in post-processing of the tests. Like filter the images/sequences within some range of results (i.e.: FP > 20%) and throw them in a folder.

If you ever deviated from the envisaged use of the system, what was the usual reason?

No

Can you think of something the system could do to help prevent this?

-

What was the hardest thing to learn about using the system?

Query syntax. I always had to read the help about that.

What feature did you particularly like?

Results history and saving/retrieving results from the server

4. Any additional comments about the system

1. Does the new interface help to speed up the task of testing image processing algorithms?

Yes it does as the automatic tests replace the manually ones which take much longer.

2. Is it easy for the user to understand how to use the system based on it's visual appearance, or are some instructions required?

I believe it is easy.

3. Does the system present any difficulties that prevent the user from carrying out tasks seamlessly?

On a very large amount of database the test is sensible slower than it could. The fact that it queries the database for a large amount of images and that the testing part is written in java makes it slower. It's just my opinion.

4. Are parts of the system irrelevant or unnecessary?

I can't think of any at the moment.

5. Overall Ratings

Please rate the system from 1 to 5 based on the following measures. Feel free to leave a comment for any of the measures.

a. Efficiency __4_____
b. User friendliness __5_____
c. Pleasant to use ____5_____
d. Easy to remember __4_____
e. Overall satisfaction__4_____
f. Potential future usage __5_____

REFERENCES

Adini, Y., Moses, Y., Ullman, S., (1997) *Face recognition: The Problem of Compensating for Changes in Illumination Direction* In IEEE Transactions on Pattern Analysis and Machine Intelligence Vol. 19 No. 7 pp.721-732.

Balasuriya L.S., Kodikara N.D., (2001) *Frontal View Human Face Detection and Recognition* In Proceedings of the International Information Technology Conference IITC2000, Colombo, January 17th-19th 2001.

Bill, R.W., (2001) *Jython for Java Programmers.* Sams Pub, December 18, 2001 ISBN-10: 0-7357-1111-9.

Black, J.A., Gargesha, M., Kahol, K., Kuchi, P., (2002) *A Framework for Performance Evaluation of Face Recognition Algorithms* In Proceedings of the International Conference on ITCOM, Internet Multimedia Systems II, Boston, July 2002.

Blackburn, D., M., Bone, J.M., Phillips, P.J., (2001). *FRVT 2000 Report* Technical Report, February 2001, Available: http://www.frvt.org/FRVT2000/documents.htm

Blackburn, D.M., (2001) *Evaluating Technology Properly – Three Easy Steps To Success* Article originally published in Corrections Today, July 2001, Vol. 63 No. 1.

Boehm, B., (1988) *A Spiral Model of Software Development and Enhancement* In IEEE Computer, vol.21, No. 5, May 1988, pp 61-72.

Bone, M., Backburn, D., (2002) *Face Recognition at a Chokepoint – Scenario Evaluation Results* Evaluation Report United States Department of Defence (Counterdrug Technology Development Program office) November 2002.

Booch, G., (1993) *Object-Oriented Analysis and Design with Applications* 2nd Edition, Addison-Wesley Professional; September 30, 1993.

Bovik, A.C., (2000) *Handbook of Image & Video Processing* Academic Press.

Bowyer, K.W., Phillips, P.J., (1998) *Overview of Work in Empirical Evaluation of Computer Vision Algorithms* In Empirical Evaluation Techniques in Computer Vision (ed. K.W.Bowyer and P.J. Phillips), IEEE Computer Society Press, Los Alamitos, 1998, pp. 1-11.

Carlson, D., (2005) *Eclipse Distilled* Addison Wesley Professional, February 14, 2005 ISBN-10: 0-321-28815-7.

Cheon, Y., Kim, M.Y., Perumandla, A., (2005) *A Complete Automation of Unit Testing for Java Programs* In Proceedings of the 2005 International Conference on Software Engineering Research and Practice (SERP '05), Las Vegas, Nevada, June 27-29, *2005,* pp. 290-295.

Chhabra, A., Phillips, I., (1998) *A Benchmark for Graphics Recognition Systems* In Empirical Evaluation Techniques in Computer Vision, Los Alamitos, CA, IEEE Computer Society Press, pp. 28-44.

Clark, A.F., Clark, C., (2002) *Performance Characterization in Computer Vision: a Tutorial* Available Online: peipa.essex.ac.uk/benchmark/tutorials/essex/tutorial.pdf

Connolly, C. (2003) *Latest Developments in Machine Vision – a Review of Image Processing Packages* In Sensor Review Vol. 23 No. 3 pp. 193 – 201. ISSN: 0260-2288 Publisher: MCB UP Ltd.

Cooper, J. W., (1998) *The Design Patterns Java Companion* Addison-Wesley, October 1998.

Courtney, P., Thacker, N.A., (2001) *Performance Characterisation in Computer Vision: The Role of Statistics in Testing and Design.* In Blanc-Talon and Popescu, editors, Imaging and Vision Systems: Theory, Assessment and Applications. NOVA Science Books, 2001.

Courtney, P., Thacker, N.A., (2003) *Performance Characterisation in Computer Vision: The Role of Statistics in Testing and Design.* August 29, 2003 Available: http://peipa.essex.ac.uk/benchmark/tutorials/epsrcss2002/handout/epsrcss2002-handout.pdf

Dalong, J., Yuxiao, H., Shuicheng, Y., Zhang, L., Zhang, H., Wen, G., (2005) *Efficient 3D Reconstruction For Face Recognition* In Pattern Recognition Letters, Vol. 38, No. 6, June 2005, pp. 787-798.

Davis, M., Van House, N.A., Burgener, C., Perkel, D., King, S., Towle, J., Ahern, S., Finn, M., Viswanathan, V., Rothenberg, M., (2005) *MMM2: Mobile Media Metadata for Media Sharing* In CHI '05 Extended Abstracts on Human Factors in Computing Systems, 2005, ACM Press, pp. 1335-1338.

Delac, K., Grgic, M., Grgic, S., (2005) *Effects of JPEG and JPEG2000 Compression on Face Recognition* In Proceedings of Third International Conference on Advances in Pattern Recognition, ICAPR 2005, LMCS 3687 Springer-Verlag Berlin Heidelberg, pp. 136-145.

Deluca, M.J., United States Patent: 6,407,777 Filed: October 9, 1997 Application No: 947603

Eastman Kodak (2006) *Infoimaging – A $255 Billion Industry Creating by the Convergence of Image Science and Information Technology* Eastman Kodak White Available online: http://www.kodak.com/go/infoimaging

Eriksson, H.-K., Penker, M., (1997) *UML Toolkit* John Wiley & Sons Inc; October 14 1997.

Firschein, O., Fischler, M.A., Kanade, T., (1993) *Creating Benchmarking Problems in Machine Vision: Scientific Challenge Problems* In Proceedings of the 1993 DARPA Image Understanding Workshop, April, 1993.

Föerstner, W., (1996) *10 Pros and Cons against Performance Characterisation of Vision Algorithms* In Proceedings of ECCV Workshop on Performance Characteristics of Vision Algorithms, Cambridge, UK, April 1996. Also in Machine Vision Applications, 9 (5/6), 1997, pp.215-218.

Gabrilovich, E., Finkelstein, L., (2001) *JNI – C++ Integration Made Easy* In C/C++ Users Journal. CMP Media LLC, Manhasset, NY, USA, January 2001.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., (1995) *Design Patterns, Elements of Reusable Object-Oriented Software* Addison Wesley (January 15, 1995) ISBN 0-201-63361-2.

Gargi, U., Deng, Y., Tretter, D.R., (2003) *Managing and Searching Personal Photo Collections* In Proc. SPIE Storage and Retrieval for Media Databases, pp. 13-21.

Geer, D., (2005) *Eclipse Becomes the Dominant Java IDE* In IEEE Computer Vol. 38 No. 7, pp. 16-18.

Girgensohn, A., Adcock, J., Cooper, M., Foote, J., Wilcox, L., (2003) *Simplifying the Management of Large Photo Collections* In INTERACT '03: Ninth IFIP TC13 International Conference on Human-Computer Interaction, IOS Press, September 2003. pp 196-203.

Girgensohn, A., Adcock, J., Wilcox, L., (2004a) *Leveraging Face Recognition Technology to Find and Organize Photos* In MIR '04: Proceedings of the 6th ACM

SIGMM international workshop on Multimedia information retrieval. ACM Press, pp 99-106.

Girgensohn, A., Adcock, J., Wilcox, L., (2004B) *Organizing Photos of People* In Proceedings of UIST 2004 Conference Companion, Santa Fe, USA. October 24, pp. 37-38.

Gonzalez, R. C., Woods, R. E., (2002) *Digital Image Processing (2nd Edition)* Prentice Hall (January 15, 2002) ISBN: 0201180758

Gould, I.D., Lewis, C., (1985) *Designing for Usability: Key Principles and What Designers Think* In Communications of the ACM Volume 28, No. 3 (Mar. 1985), pp. 300-311.

Green, D., DiCaterino, A., (1998) *A Survey of System Development Process Models* Center for Technology in Government, University at Albany/SUNY, CTG.MFA – 003, February 1998, Available online: http://www.ctg.albany.edu/publications/reports/survey_of_sysdev

Grgic, M., Delac, K., Grgic, S., (2005) *Face Recognition: Hypothesis Testing Across All Ranks* Technical Report, FER-VCL-TR-2005-02, University of Zagreb, FER.

Gross, R., Baker, S., Matthews, I., Kanade T., (2004) *Face Recognition across Pose and Illumination* In Handbook of Face Recognition, (Stan Z. Li and Anil K. Jain, ed.,) Springer-Verlag, June, 2004.

Gross, R., Cohn, J., Shi, J., (2001) *Quo Vadis Face Recognition* Third Workshop on Empirical Evaluation Methods in Computer Vision, In IEEE Conference on Computer Vision and Pattern Recognition, Hawaii, December 2001.

Gruber, O., Hargrave, B.J., McAffer, J., Rapicault, P., Watson, T., (2005) *The Eclipse 3.0 platform: Adopting OSGi technology* In IBM SYSTEMS JOURNAL, Volume 44, No 2, 2005 pp.289-299.

Gur, H., (2002) *Newgen Software Technologies* Available Online: http://www.expresscomputeronline.com/20020408/technology2.shtml 8 Apr 2002 (Accessed 19 Jan 2005)

Hailpern, B., Santhanam, P., (2002). *Software Debugging, Testing, and Verification* In IBM Systems Journal, Volume 41, Number 1, pp. 4-12.

Harold, E.R., Means, W.S., (2004) *XML in a Nutshell*. 3rd Edition, O'Reilly, September 2004, ISBN: 0-596-00764-7.

Hatton, T., (2005) *SWT: A Developers Notebook* O Reilly Media Inc., October 2004.

Heo, J., Abidi, B., Paik, J., Abidi, M.A., (2003) *Face recognition: Evaluation Report for FaceIt Identification and Surveillance* .In Proc. Of SPIE 6th International Conference on Quality Control by Artificial Vision, Vol. 5132, Gatlinburg, TN, May 2003, pp. 551-558.

Heseltine, T., Pears, N., Austin, J., Chen, Z., (2003) *Face Recognition: A Comparison of Appearance-Based Approaches* In Proc. 7th Digital Image Computing: Techniques and Applications, (Sun C., Talbot H., Ourselin S. and Adriaansen T. (Eds.)), Dec. 2003, Vol. 1 pp. 59-69.

Hong, J.-H., Yun, E.-K., Cho, S.-B., (2004) *A Review of Performance Evaluation for Biometrics System* In International Journal of Image and Graphics World Scientific Publishing Company March 24, 2004.

Hsu, R.L., Abdel-Mottaleb, M., Jain, A.K., (2001) *Face Detection in Color Images* In Proc. International Conference Image Processing, pp. 1046-1049.

http://jakarta.apache.org/bsf/ (2006) The Apache Jakarta Project- Bean Scripting Framework (Accessed 20 April 2006)

http://jruby.sourceforge.net/ (2006) (Accessed 20 April 2006)

http://msdn.microsoft.com/visualc/ (2006) (Accessed 20 April 2006)

http://news.bbc.co.uk/1/hi/in_depth/uk/2005/london_explosions/default.stm, (2005) (Accessed 20 April 2006)

http://peipa.essex.ac.uk/ PEIPA, the Pilot European Image Processing Archive (Accessed 05 May 2006)

http://java.com/en/about/, 2006 Sun Microsystems, Inc. (Accessed 20 May 2006)

http://www.apple.com (2006) Apple Computer, Inc. (Accessed 20 May 2006)

http://www.beanshell.org/ (2006) (Accessed 17 May 2006)

http://www.bell-labs.com/ (2006) Bell Laboratories (Accessed 20 May 2006)

http://www.biometricgroup.com/ (2006) International Biometric Group (Accessed 17 May 2006)

http://www.eclipse.org/ (2006) (Accessed 20 May 2006)

http://www.eclipse.org/swt (2006) The Standard Widget Toolkit (Accessed 20 May 2006)

http://www.ge.com/ (2006) General Electric (Accessed 20 May 2006)

http://www.jpeg.org/ (2005) (Accessed 24 January 2005)

http://www.junit.org (2006) (Accessed 20 April 2006)

http://www.jython.org (2006) (Accessed 20 April 2006)

http://www.python.org (2006) (Accessed 20 April 2006)

http://www.mozilla.org/rhino/ (2006) Rhino: JavaScript for Java (Accessed 20 April 2006)

http://www.september11news.com/, (2005) (Accessed 20 April 2006)

Hua, X.-S., Wenyin, L., Zhang, H.-J., (2004) *An Automatic Performance Evaluation Protocol for Video Text Detection Algorithms* In IEEE Transactions on circuits and systems for video technology, Vol. 14 No. 4, April 2004, pp. 498-507.

Hunter, D., Gibbons, D., Ozu, N., Pinnock, J., Cagle, K., (2000) *Beginning Xml (Programmer to Programmer)* Peer Information Inc.; 1st edition, (June 1, 2000) ISBN 1861003412.

International Data Corporation, (2004) Worldwide Image Forecast, 2004-2008: The Image Bible (IDC #32428), December 2004, Available Online: www.idc.com

Jähne, B., (1997) Practical Handbook on Image Processing for Scientific and Technical Applications. CRC Press 1997, ISBN 0849389062 (alk. paper)

Jain, A. K., Ross, A., Prabhaka, S., (2004) *An Introduction to Biometric Recognition* In IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Image and Video-Based Biometrics. Vol. 14 No. 1, pp. 4–20.

Jaynes, C., Kale, A., Sanders, N., Grossmann, E., (2005) *The Terrascope Dataset: A Scripted Multi-Camera Indoor Video SurveillanceDataset with Ground-truth* In Proceedings of the IEEE Workshop on Visual Surveillance and Performance Analysis for Tracking and Surveillance.

Kendall, K.E., Kendall, J.E., (2001) *Systems Analysis and Design* 5th Edition Prentice Hall, June 2001.

Kim, Y.-O., Jang, S.-H., Park, C.-W., Sung, H.-G., Kwon, O., Paik, J., (2004) *A New 3D Active Camera System for Robust Face Recognition by Correcting Pose Variation* In Proceedings of International Conference on Control, Automation and Systems (ICCAS 2004) August 2004 pp. 1482-1487.

Kim, Y.-O., Paik, J., Heo, J., Koschan, A., Abidi, B., Abidi, M., (2003) *Automatic Face Region Tracking for Highly Accurate Face Recognition in Unconstrained Environments* In Proc. IEEE International Conference on Advanced Video and Signal Based Surveillance, Miami, FL, July 2003. pp. 29-36.

Knudsen, J., Niemeyer, P., (2005) *Learning Java* 3rd Edition, O'Reilly, May 2005 ISBN: 0-596-00873-2

Kong, S.G., Heo, J., Abidi, B.R., Paik, J., Abidi, M.A., (2005) *Recent Advances in Visual and Infrared Face Recognition – A Review* In the Journal of Computer Vision and Image Understanding, Vol. 97 No. 1, June 2005, pp.103-135.

Kung, S.Y., Mak, M.K., Lin, S.H., (2004) *Biometric Authentication: A Machine Learning Approach* Prentice Hall PTR, September 14, 2004, Print ISBN-10: 0-13-147824-9.

Liang, S., (1999) *Java™ Native Interface: Programmer's Guide and Specification* Addison Wesley Professional, June 10, 1999 ISBN-10: 0-201-32577-2.

Little, G., Krishna, S., Black, J., Panchanathan, S., (2005) *A Methodology for Evaluating Robustness of Face Recognition Algorithms with Respect to Variations in Pose Angle and Illumination Angle* In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2005), September pp. 89-92.

Liu, W., Dori, D., (1999) *Principles of Constructing a Performance Evaluation Protocol for Graphics Recognition Algorithms* In Performance Characterization and Evaluation of Computer Vision Algorithms, R. Klette, S. Stiehl, and M. Viergever (eds.), Kluwer, pp. 97-106.

Low, B.K., Hjelmas, E., (2001) *Face Detection: A Survey* Computer Vision and Image Understanding, Sept. 2001, Vol. 3 No. 3, pp. 236-274.

Loy, G., Eklundh, J., (2005) *A Review of Benchmarking Content Based Image Retrieval* In Proceedings of the MUSCLE / ImageCLEF Workshop on Image and Video Retrieval Evaluation, September 2005.

Lu, X., (2003) *Image Analysis for Face Recognition*, personal notes, May 2003, Available Online: http://www.cse.msu.edu/%7Elvxiaogu/publications/ImAna4FacRcg_Lu.pdf

Luo, H., Yen, J., Tretter, D., (2004) *An Efficient Automatic Redeye Detection and Correction Algorithm* 17th International Conference on Pattern Recognition (ICPR'04). Cambridge, UK. Vol. 2, pp. 883-886.

Lyra Research (2006) *The Big Picture: What Happens to Photo Prints?* The 2006 Lyra Imaging Symposium, 02 March 2006.

Mancuso, M., Battiato, S., (2001) *An Introduction to the Digital Still Camera Technology* In ST Journal of System Research, Vol.2 No.2, Dec.2001.

Mansfield, A.J., Wayman, J.L., (2002) *Best Practices in Testing and Reporting Performance of Biometric Devices* Issue 2 draft 9, Issued 08/02, United Kingdom Government Biometric Working Group. Available Online: http://www.npl.co.uk/scientific_software/publications/biometrics/bestprac_v2_1.pdf

Martin, A., Doddington, G., Kamm, T., Ordowski, M., and Przybocki, M., (1997) *The DET Curve In Assessment of Detection Task Performance* In Proceedings of the European Conference on Speech Communication and Technology, 1997, pp. 1895-1898.

Maruyama, H., Tamura, K., Uramoto, N., (1999) *XML and Java Developing Web Applications* Addison-Wesley Professional, May 04, 1999 ISBN-10: 0-201-48543-5.

Mazor, B., (2005) *Biometric Imaging Faces a Reality Check* Discussion Featuring Biometric Imaging and Security Concerns. In Advanced Imaging Magazine, September 2005, Available Online: http://www.advancedimagingpro.com/publication/article.jsp?pubId=1&id=1743

McAffer, J., Lemieux, J.-M., (2005) *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java™ Applications* Addison Wesley Professional, October 11, 2005 ISBN-10: 0-321-33461-2.

McGregor, J.D., Sykes, D.A., (2001) *A Practical Guide to Testing Object-Oriented Software* Addison Wesley Professional, March 05, 2001 ISBN-10: 0-201-32564-0

McLaughlin, B., (2000) *Java & XML* O' Reilly Publishing, June 2000, ISBN: 0-596-00016-2.

McLaughlin, B., (2001) *Java & XML* 2nd Edition, O' Reilly Publishing, September 2001, ISBN: 0-596-00197-5.

Meer, P., Stewart, C.V., Tyler, D.E., (2000) *Introduction – Robust Computer Vision: An Interdisciplinary Challenge* In Computer Vision and Image Understanding 78 Published by Academic Press, pp. 1-7.

Messina, G., Castorina, A., Battiato, S., Bosco, A., (2003) *Image Quality Improvement by Adaptive Exposure Correction Techniques* In Proc. of ICME 2003. pp. 549-552.

Micheals, R.J., Boult, T.E., (2000) *Replicate Statistics for Efficient Vision Systems* Technical report (Lehigh University) December 2000 Available Online: http://www.eecs.lehigh.edu/

Micheals, R.J., Boult, T.E., (2001) *Efficient Evaluation of Classification and Recognition Systems* In IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (CVPR 2001), vol. 1, Dec. 2001, pp. I50–I57.

Milburn, K., (2004) *Digital Photography: Expert Techniques* O'Reilly Publishing, March 2004 ISBN: 0-596-00547-4.

MIT face database, (2000) Massachusetts Institute of Technology. Available online: ftp://whitechapel.media.mit.edu/pub/images/

Moon, H., Chellapa, R., Rosenfeld, A., (2002) *Performance Analysis of a Simple Vehicle Detection Algorithm* In Image and Vision Computing, Vol. 20 No. 1, pp. 1–13.

Mordani, R., Davidson, J.D., Boag, S., (2001) *Java API for XML Processing* Sun Microsystems, Inc. Version 1.1 Final Release, February 6, 2001.

Mu, X., Artiklar, M., Artiklar, M., Hassoun, M.H., Watta, P., (2001) *Training Algorithms for Robust Face Recognition Using a Template-matching Approach* In Proc. International Joint Conference on Neural Networks (IJCNN '01), Vol. 4, Washington, DC, USA, July 2001, pp. 2877–2882.

Müller, H., Geissbuhler, A., Marchand-Maillet, S., Clough, P., (2004) *Benchmarking Image Retrieval Applications* In Proceedings of The Tenth International Conference on Distributed Multimedia Systems (DMS'2004), Workshop on Visual Information Systems (VIS 2004), San Francisco, CA, USA, 2004.

Narasimhan, S.G., Wang, C., Nayar, S. K., (2002) *All the Images of an Outdoor Scene* In Proc. ECCV, 2002. LNCS 2352, 2002, pp. 148-162.

Narayanaswami, C., Raghunath, M.T., (2004) *Expanding the Digital Camera's Reach* In IEEE Computer, Vol. 37 No. 12, Dec 2004, pp. 65-73.

Nishino, K., Belhumeur, P.N., Nayar, S.K., (2005a) *Using Eye Reflections for Face Recognition Under Varying Illumination* In Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1, pp. 519-526.

Nuseibeh, BA., (2001) *Weaving Together Requirements and Architecture* In IEEE Computer Volume 34 No. 3, pp.115-117 (March 2001).

Ojala, T. Mäenpää, T. Pietikäinen, M. Viertola, J. Kyllönen, J. Huovinen, S. (2002) *Outex - New Framework for Empirical Evaluation of Texture Analysis Algorithms* In *Proc. 16th International Conference on Pattern Recognition*, Vol. 1, Quebec, Canada, 2002, pp. 701–706.

Orfali, R., Harkey, D., Edwards, J., (1999) *Client/Server Survival Guide* Third Edition, John Wiley & Sons, 1999 ISBN 0-471-31615-6.

Pankanti, S., Bolle, R.M., Jain, A., *Guest Editors' Introduction: Biometrics - The Future of Identification* In IEEE Computer, Vol. 33, No. 2, pp. 46-49.

Pedroni, S., Rappin N., (2002) Jython Essentials: Rapid Scripting in Java 1st Edition, O'Reilly & Associates, ISBN:0596002475.

Pentland, A., Choudhury, T., (2000) *Face Recognition for Smart Environments* In IEEE Computer Magazine (Special issue on biometrics), Vol. 33, No. 2. (February 2000), pp. 50-55.

Phillips, I.T., Chhabra, A.K., (1999) *Empirical Performance Evaluation of Graphics Recognition Systems* In IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 21 No. 9, September 1999.

Phillips, I.T., Liang, J., Chhabra, A.K., Haralick, R., (1998) *A Performance Evaluation Protocol for Graphics Recognition Systems.* In Graphics Recognition: Algorithms and Systems, Second International Workshop, GREC'97, Nancy, France, August 1997, Selected Papers, Vol. 1389 of Lecture Notes in Computer Science, pp. (K. Tombre and A. Chhabra, Eds), Springer, Berlin, 1998, pp.372–389.

Phillips, P.J., Flynn, P.J., Scruggs, T., Bowyer, K.W., Chang, J., Hoffman, K., Marques, J., Min, J., Worek, W., (2005) *Overview of the Face Recognition Grand Challenge* Computer Vision and Pattern Recognition (CVPR 2005), San Diego, June 2005, pp. 947-954.

Phillips, P.J., Grother, P., Micheals, R.J., Blackburn, D.M., Tabassi, E., Bone, J.M., (2003) *FRVT 2002: Evaluation report* Tech. Rep., March 2003, Available online: http://www.frvt.org/FRVT2002/documents.htm.

Phillips, P.J., Martin, A., Wilson, C.L., Przybocki, M., (2000a) *An Introduction to Evaluating Biometric Systems* In IEEE Computer Magazine (Special issue on biometrics), Vol. 33, No. 2 (February 2000), pp. 56-63.

Phillips, P.J., Moon, H., Rizvi, S.A., Rauss, P.J., (2000b) *The FERET Evaluation Methodology for Face-Recognition Algorithms* In IEEE Transactions on Pattern Analysis and Machine Intelligence, October 2000, Vol. 22 No. 10.

Pluta, J., (2004) *Eclipse: Step by Step. A Practical Guide to Becoming Proficient in Eclipse* MC Press Online, October 2004.

Prabhakar, S., Pankanti, S., Jain, A.K., (2003) *Biometric Recognition: Security and Privacy Concern* In IEEE Security and Privacy Magazine, Vol. 1, No. 2, pp. 33-42.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey, T., (1994) *Human-Computer Interaction : Concepts And Design* (Hardcover) Addison Wesley; 1st edition (April 30, 1994) ISBN: 0201627698.

Pressman R. S., (2005) *Software Engineering A Practitioner's Approach* 6th Edition, McGraw-Hill ISBN: 0007-123840-9.

Raskin, J., (2000) *The Humane Interface: New Directions for Designing Interactive Systems* Addison Wesley, April 6 2000, ISBN: 0201379376.

Riopka, T., Boult, T., (2003) *The Eyes Have It* In *ACM Workshop on Biometric Methods & Application.* (WBMA'03), November 2003, Berkley, pp. 9-16.

Sarvas, R., (2005) *User-centric Metadata for Mobile Photos* In Proc. Pervasive Image Capture and Sharing Workshop (PICS2005) at Ubicomp 2005.

Sarvas, R., Herrarte, E., Wilhelm, A., Davis, M., (2004) *Metadata Creation System for Mobile Images* In Proceedings of MobiSys 2004, Boston, MA, USA. ACM Press, New York, NY, 2004, pp. 36-48.

Schettini, R., Gasparini, F., Chazli, F., (2004) *A Modular Procedure for Automatic Red Eye Correction in Digital Photos* In Proceedings of SPIE. Vol. 5293 (R. Eschbach, G.G. Marcu eds.), pp.139-147.

Shah, M., (2002) *Guest Introduction: The Changing Shape of Computer Vision in the Twenty-First Century* In International Journal of Computer Vision, November 2002

Shalloway, A., Trott, J.R., (2004) *Design Patterns Explained A New Perspective on Object-Oriented Design* Second Edition, Addison Wesley Professional, Pub Date: October 12 2004, ISBN-10: 0-321-24714-0.

Sharma, P., Reilly, R.B., (2003) *A Colour Face Image Database for Benchmarking of Automatic Face Detection Algorithms* In 4th EURASIP Conference, Video/Image Processing and Multimedia Communications, 2003, 2-5 July 2003.

Shavor, S., D'Anjou, J., Fairbrother, S., Kehn, D., Kellermen, J., McCarthy, P., (2003) *The Java Developer's Guide to ECLIPSE* Addison-Wesley, June 2003.

Smolka, B., Czubin, K., Hardeberg, J.Y., Plataniotis, K.N., Szczepanski, M., Wojciechowski, K., (2003) *Towards Automatic Redeye Effect Removal* In Pattern Recognition Letters Vol. 24 No. 11. July 2003, pp. 1767-1785.

Sommerville, I., (1992) *Software Engineering* 4th Edition, Addison-Wesley Publishing Company.

Sommerville, I., (2001) *Software Engineering* 6th Edition, Addison-Wesley Publishing Company.

Takeuchi, A., Shneier, M., Hong, T., Chang, T., Scrapper, C., Cheok, G., (2003) *Ground Truth and Benchmarks for Performance Evaluation* In Proceedings of SPIE-the International Society For Optical Engineering, Vol. 5083 No. 48, April 2003, pp. 408-414.

Thacker, N., Lacey, T., Courtney, P., (2003) *An Empirical Design Methodology for the Construction of Machine Vision Systems* Tina memo 2002-005, Technical report, Department of Imaging Science and Biomedical Engineering, Medical School, University of Manchester, UK, May 2003. Available online: http://peipa.essex.ac.uk/benchmark/methodology/white-paper/methodology.pdf

Thacker, N.A., Clark, A.F., Barron, J., Beveridge, R., Clark, C., Courtney, P., Crum, W.R., Ramesh, V., (2005) *Performance Characterisation in Computer Vision: A Guide to Best Practices* Tina memo 2005-009, Technical report, Department of Imaging Science and Biomedical Engineering, Medical School, University of Manchester, UK, April 2005. Available online: http://www.tina-vision.net/docs/memos/2005-009.pdf

Torres, L., (2004) *Is There Any Hope for Face Recognition?* In Proc. of the 5th International Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS 2004, 21-23 April 2004, Lisboa, Portugal.

Ulichney, R., Gaubatz, M., Thong, JM. V. P., (2003) *RedBot- A Tool for Improving Red-Eye Correction.* Available online: http://hpl.hp.com/research/redbot/. October 2003.

Umbaugh, S. E., (1998) *Computer Vision and Image Processing: A Practical Approach using CVIPtools* Prentice Hall PTR Publishing

Van House, N., Davis, M., Ames, M., Finn, M., Viswanathan, V., (2005) *The Uses of Personal Networked Digital Imaging: An Empirical Study of Cameraphone Photos and Sharing* In CHI '05 Extended Abstracts on Human Factors in Computing Systems, 2005, ACM Press, pp. 1853-1856.

Van House, N., Davis, M., Takhteyev, Y., Good, N., Wilhelm, A., Finn, M., (2004 a) *From 'What?' to 'Why?': The Social Uses of Personal Photos* In CSCW'04, November 6–10, 2004, Chicago, Illinois, USA. ACM Press 2004.

Vaughan-Nichols, S.J., (2003b) *XML Raises Concerns As It Gains Prominence* In IEEE Computer Volume 36, Issue 5, May 2003 pp. 14 – 16. Vol. 50, No. 2, pp. 103-110.

Werbicki, P., Kremer, R., (2005) *Object-Oriented Programming across Native-Virtual Boundaries* Virtual Execution Environments (Vee'05), June 11-12, 2005, Chicago, Illinois, USA. ACM.

Whittaker, J.A., (2000) *What is Software Testing? And Why is It So Hard?* In IEEE Software Volume 17, No. 1 (Jan 2000), pp.70-79.

Wilhelm, A., Takhteyev, Y., Sarvas, R., Van House, N., Davis, M., (2004) *Photo Annotation on a Camera Phone* In Proc. of CHI2004, ACM Press, 2004, pp. 1403-1406.

Yang, C-H.T., Lai, S.-H., Chang, L.-W., (2004) *Robust Face Image Matching Under Illumination Variations* In EURASIP Journal on Applied Signal Processing Vol. 16 Hindawi Publishing Corporation, pp. 2533-2543.

Yang, M.-H., Kriegman, D., Ahuja, N., (2002) *Detecting Faces in Images: A Survey* In IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24 No. 1, pp. 34-58.

Yang, Z., Zage, D., Zage, W., (2005) *The Eclipse Platform for Tool Integration and Development* SERC#: SERC-TR-273 Publication Date: 5/1/2005 Software Engineering Research Center, An NSF Industry/University Cooperative Research Center.

Young, M., (2000) *Step by Step XML* Microsoft Press, July 2000.

Young, M.L., (2002) *Internet: the Complete Reference* Osborne/McGraw-Hill; 2nd edition (June 6, 2002) ISBN: 0072194154.

Zhang, L., Chen, L., Li, M., Zhang, H., (2003) *Automated Annotation of Human Faces in Family Albums* In Proceedings of the 11th International Conference on Multimedia (MM2003), ACM Press, 2003, pp.335-358.

Zhang, L., Hu, Y., Li, M., Ma, W., Zhang, H., (2004a) *Efficient Propagation for Face Annotation in Family Albums* In Proceedings of the 12th International Conference on Multimedia (MM2004), ACM Press, pp. 716-723.

Zhang, L., Li, M., Zhang, H.-J., (2002) *Boosting Image Orientation Detection with Indoor vs. Outdoor Classification* In Proceedings of IEEE Workshop on Applications of Computer Vision, pp. 95-99.

Zhang, L., Sun, Y., Li, M., Zhang, H., (2004b) *Automated Red-Eye Detection and Correction in Digital Photographs* In Proc. IEEE International Conference on Image Processing (ICIP 2004). Singapore, October 24-27, Vol. 4 pp. 2363 – 2366.

Zhao, W.Y., Chellappa, R., (2002) *Image-based Face Recognition: Issues and Methods* Image Recognition and Classification, (Ed. B. Javidi, M. Dekker) 2002, pp. 375-402.

Zhao, W., Chellappa, R., Rosenfeld, A., Phillips, P.J., (2000) *Face recognition: A Literature Survey* Technical Report, CAR-TR-948, University of Maryland, College Park, 2000.

Zhao, W., Chellappa, R., Rosenfeld, A., Phillips, P.J., (2003) *Face recognition: A Literature Survey.* In ACM Computing Surveys, 2003, pp. 399-458.