

A Bioinformatics Framework for the Management and Analysis of High Throughput CGH Microarray Projects

James Anthony Morris BSc(Hons), MRes

UCL EGA Institute for Women's Health

Thesis submitted for the degree of PhD in Bioinformatics

Declaration

I, James Anthony Morris confirm that the work presented in this thesis is my own.
Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Dedication

This thesis is dedicated to those in my life whose love and support made it possible for me to pursue and complete this work.

To my Father, my Mother and my Wife

Acknowledgements

- My first acknowledgement needs to go to my primary supervisor Chris Jones, I would like to say a really big thank to him for all his help, support and guidance all the way through this project.
- I would like to thank Tanya Lebi for all her hard work in performing all the aCGH experiments for the MALOVA study and for also being our main customer during the development of the majority of the software in this project.
- As my secondary supervisor Simon Gayther provided very useful support and guidance at key points in this project, I would like to thank him for his help and for allowing me to work in his laboratory.
- I would like to thank the Eve Appeal and Mermaid charities for their support of my project.
- I would finally like to thank Jeff Barrett, as my current boss, he has encouraged and enabled me to be able to complete this thesis.

Abstract

High throughput experimental techniques have revolutionised biological research; these techniques enable researchers, in an unbiased fashion to survey entire biological systems such as all the somatic mutations in a tumour in a single experiment. Due to the often complex informatics demands of these techniques, robust computational solutions are required to ensure high quality reproducible results are generated. The challenge of this thesis was to develop such a computational solution for the management and analysis of high throughput microarray Comparative Genomic Hybridisation (aCGH) projects. This task also provided an opportunity to test the hypothesis that agile software development approaches are well suited for bioinformatics projects and that formalised development practices produce better quality software. This is an important question as formalised software development practices have been underused so far in the field of bioinformatics.

This thesis describes the development and application of a bioinformatics framework for the management and analysis of microarray CGH projects. The framework includes: a Laboratory Information Management System (LIMS) that manages and records all aspects of microarray CGH experimentation; a set of easy to use visualisation tools for aCGH experimental data; and a suite of object oriented Perl modules providing a flexible way to construct data pipelines quickly using the statistical programming language R for quality control, normalisation and analysis. In order to test the framework, it was successfully applied in the aCGH profiling of 94 ovarian tumour samples. Subsequent analysis of these data identified 4 well supported genomic regions which appear to influence patient survival.

The evaluation of agile practices implemented in this thesis has demonstrated that they are well suited to the development of bioinformatics solutions as they

enable developers to react to the changes of this rapidly evolving field, to create successful software solutions such as the bioinformatics framework presented here.

Contents

List of Figures	xi
List of Tables	xiii
1 Introduction	1
1.1 Genetic alterations and cancer	1
1.1.1 Ovarian cancer	2
1.2 High throughput technologies	3
1.2.1 Background	3
1.2.2 High throughput technology challenges	4
1.3 Microarrays	6
1.3.1 Array comparative genomic hybridisation	6
1.3.2 Background	7
1.3.3 aCGH informatics	10
1.3.4 Downstream analysis	14
1.3.5 Laboratory information management systems	21

1.4	Software development for high throughput science	24
1.4.1	Software development processes	26
1.4.2	Software development best practices	34
1.4.3	Programming languages	41
1.4.4	Bioinformatics software development	41
1.4.5	Bioinformatics software development review	43
1.5	Aims	50
2	LIMS	52
2.1	Background	52
2.1.1	The existing in house LIMS	53
2.1.2	Initial ArrayPipeLine LIMS code base	54
2.1.3	Extending the LIMS database for biological samples	55
2.1.4	Extending the LIMS database for aCGH experiments	58
2.1.5	Extending the LIMS database for recording analysis steps	61
2.1.6	Extending the ArrayPipeLine LIMS GUI	62
2.2	Methods and implementation	63
2.2.1	Database	64
2.2.2	Creating an object oriented Perl API	68
2.2.3	User interface layer	68
2.2.4	Agile software development	69
2.3	Results	69
2.3.1	Requirements gathering	69
2.3.2	Object orientation of the ArrayPipeLine LIMS source code	71
2.3.3	Extending the ArrayPipeLine LIMS database for sample data	77

2.3.4	Extending the ArrayPipeLine LIMS database for aCGH experiment data	78
2.3.5	Extending the ArrayPipeLine LIMS database for analysis pipeline data	82
2.3.6	Extending the ArrayPipeLine GUI	85
2.3.7	Continuous integration	86
2.4	Conclusions	90
3	An analysis pipeline for aCGH	95
3.1	Background	95
3.2	Implementation	98
3.2.1	Technical requirements of an analysis pipeline	98
3.2.2	Integrating Perl and R	100
3.2.3	User requirements of an analysis pipeline	105
3.2.4	Pipeline construction	106
3.2.5	Development techniques	107
3.3	Results	108
3.3.1	Analysis pipeline	108
3.3.2	Visualisation of results	117
3.3.3	Testing	127
3.3.4	Customer feedback	128
3.4	Conclusions	132
4	MALOVA	138
4.1	Background	138
4.2	Materials and methods	139
4.2.1	Patient data	139

4.2.2	Sample preparation	140
4.2.3	The Mermaid CGH microarray	140
4.2.4	Microarray hybridisation	141
4.2.5	Data analysis	142
4.2.6	Expert supervised filtering	142
4.2.7	Statistical analysis	143
4.3	Results	151
4.3.1	Quality control	151
4.3.2	Recurrent region identification	152
4.3.3	Significant recurrent copy number aberrations in stage III/IV serious ovarian tumours	153
4.3.4	Significant recurrent copy number aberrations in good prognosis tumours	154
4.3.5	Significant recurrent copy number aberrations in poor prognosis tumours	154
4.3.6	Unique Breakpoint Identification	155
4.3.7	Recurrent region feature selection	155
4.3.8	SVM Classifier Validation	161
4.3.9	Most informative regions	161
4.3.10	Gene Analysis	164
4.3.11	Random forests	165
4.4	Conclusions	167
5	Discussion	168
5.1	Extension of the ArrayPipeLine LIMS	172
5.2	Analysis pipeline	173

5.3	MALOVA	176
5.4	Bioinformatics software development practices	178
5.5	Future prospects	181
5.6	Concluding remarks	183
Bibliography		185
Appendices		204
.1	Source code appendix	204
.1.1	Chapter 2	205
.1.2	Chapter 3	206
.1.3	Chapter 4	207

--- List of Figures

1.1	Microarray CGH experiment workflow	9
1.2	Microarray CGH informatics data transformation steps	12
1.3	Support vector machine	20
2.1	Entity Relationship Diagram (ERD) of the sample information portion of the ArrayPipeLine LIMS	79
2.2	Entity Relationship Diagram (ERD) of the aCGH experiment recording portion of the ArrayPipeLine LIMS	83
2.3	Entity Relationship Diagram (ERD) of the data analysis portion of the ArrayPipeLine LIMS	84
2.4	Screenshot of the ArrayPipeLine LIMS new tumour interface	86
2.5	Screenshot of the ArrayPipeLine LIMS interface for recording labelling experiments	87
2.6	Screenshots of the ArrayPipeLine samples interface.	88
2.7	Screenshots of the ArrayPipeLine external sample interface.	89
2.8	Screenshots of the ArrayPipeLine administration interface.	91

2.9	Continuous integration of source code.	93
3.1	Quality control and data processing of experiment data	119
3.2	MA plots	121
3.3	Intensity scatter plots	122
3.4	Log ₂ intensity ratio heatmaps	124
3.5	Single channel intensity heatmaps	126
3.6	CGH plots	129
4.1	Expert supervised filtering examples	144
4.2	KCSmart profile from all samples	156
4.3	KCSmart profile from all samples	157
4.4	KCSmart profile from all samples	158

List of Tables

4.1	KC-Smart Regions Identified	152
4.2	KC-Smart features selected for classification	160
4.3	Best performing combination of KC-Smart features	161
4.4	Recurrent regions ranked by contribution (unique segments and CGHcall calls)	162
4.5	Recurrent regions ranked by contribution (BAC log2 ratios)	163
4.6	Recurrent regions ranked by contribution (unique segments and CGHcall calls)	164
4.7	Genes contained in the most influential classification regions	166
5.1	Regions of overlap with Etemadmoghadam <i>et al</i>	177

List of abbreviations

aCGH	Microarray Comparative Genomic Hybridisation.
API	Application Programming Interface.
BAC	Bacterial Artificial Chromosome.
BLOB	Binary Long OBject.
BPRC	BACPAC Resource Centre.
CGH	Comparative Genomic Hybridisation.
CGI	Common Gateway Interface.
CHORI	Children's Hospital Oakland Research Institute.
CPAN	Comprehensive Perl Archive Network.
CRAN	Comprehensive R Archive Network.
CVS	Concurrent Version Control.

DBI	DataBase Interface.
DSDM	Dynamic Systems Development Method.
EC2	Elastic Compute Cloud.
ER	Entity Relationship.
ERD	Entity Relationship Diagram.
FIGO	International Federation of Gynecology and Obstetrics.
FPPE	Formalin Fixed Paraffin Embedded.
GAL	Genepix Array Layout.
GUI	Graphical User Interface.
HG17/18	Human Genome Build 17/18.
HMM	Hidden Markov Model.
HTML	Hyper Text Markup Language.
IO	Input/Output.
KSE	Kernel Smoothed Estimate.
LIMS	Laboratory Information Management System.
MALOVA	MALignant OVArrian cancer.
Mb	Mega Base.
OO	Object Oriented.
ORM	Object Relational Mapping.

PBL	Peripheral Blood Lymphocyte.
QC	Quality Control.
RDBMS	Relational DataBase Management Software.
SQL	Structured Query Language.
TIFF	Tagged Image File Format.
TRL	Translational Research Laboratory.
UCSC	University California Santa Cruz.
UI	User Interface.
URL	Universal Resource Locator.
VCS	Version Control Software.
XP	eXtreme Programming.

CHAPTER 1

Introduction

1.1 Genetic alterations and cancer

Cancer is a disease characterised as the growth of malignant neoplasms or tumours, which are caused by the uncontrolled growth of a group of cells. All tumours are the result of accumulation of DNA alterations (Stratton *et al.*, 2009). With tumour development being driven by the combined processes of genetic instability and selection, resulting in clonal expansion of cells that have accumulated the most advantageous set of genetic aberrations (Pinkel and Albertson, 2005). These genetic changes drive a stepwise process which enables normal somatic cells to escape their normal function in the tissue and become self-sufficient in survival. The processes that must be deregulated to enter such a neoplastic state are: cell signalling, programmed cell death, anchorage dependant cell death, motility, adhesion and invasion, and attracting stromal components, including mesenchymal stem cells to bring about the formation of new blood vessels (Bast *et al.*, 2009).

The genetic aberrations found in cancer genomes have come about through a

number of different mechanisms including substitutions of one base by another; insertions or deletions of small or large segments of DNA; rearrangements, in which DNA has been broken and then rejoined to a DNA segment from elsewhere in the genome; copy number increases from the two copies present in the normal diploid genome, sometimes to several hundred copies (known as gene amplification); and copy number reductions that may result in complete absence of a DNA sequence from the cancer genome.

The identification and mapping of the genetic alterations found in tumours help researchers in a number of ways, they can point towards the genes involved in cancer progression as the chromosomal aberrations reflect oncogene activation and loss of tumour suppressor genes. Thus offering a basis for better understanding of cancer development and more importantly providing improved tools for clinical management of cancer, such as new diagnostics and therapeutic targets (Kallioniemi, 2008).

1.1.1 Ovarian cancer

Ovarian cancer has a very poor survival rate, with about 65% of diagnosed women dying within five years (Levi, 1999). The majority of ovarian cancers are thought to arise from the cells that cover the ovarian surface, but the cellular origins of the disease are still not widely agreed upon. There are currently no effective biomarkers proven able to identify early-stage disease and no reliable prognostic markers for predicting clinical response and guiding treatment (Lawrenson and Gayther, 2009). Improvements in the survival rate of ovarian cancer are very difficult because the disease poses a number of problems for researchers and clinicians. These problems include the heterogeneity of the cancers; the disease is in fact a group of tumours (serous, endometrioid, mucinous and clear cell) each of which have different clinical

features and are likely to have different genetic backgrounds (Bast *et al.*, 2009). Recognised symptoms of the disease are vague, and as a result the majority of cases are diagnosed at a late stage of disease. However, it is hoped that the use of new genetic analysis tools, such as aCGH, might help to identify new therapeutic targets that lead to improvements in the long term survival of ovarian cancer patients.

1.2 High throughput technologies

1.2.1 Background

High throughput biological technologies are experimental processes that allow the simultaneous measurement of many thousands of analytes during a single experiment, coupled with the capability to process very large numbers of specimens. Ever since their first emergence in the late 90s, these techniques have become extremely popular in biological research and the use of high throughput technologies is now widespread. The main advantage of high throughput approaches is that, by applying these technologies, researchers are able to capture a more complete picture of the system under study, such as all the somatic mutations across a genome, or the transcription levels of all the genes in the transcriptome of a cell. This is instead of just being able to capture a snapshot of information regarding a single part of the system, as is the case using low throughput approaches. The ability to assay the system under study completely enables research to be carried out in a hypothesis agnostic fashion. This is a major reason behind their widespread adoption, instead of being driven by previous findings and assumptions using a hypothesis driven approach. This gives researchers the opportunity to make truly novel discoveries that would be otherwise impossible to detect. This has been, and will continue to

be, very important in advancing our knowledge of biology.

1.2.2 High throughput technology challenges

The greatest challenge in the application of high throughput technologies to answer biological questions comes in the form of the informatics requirements which are created by the large scale nature of these experiments. Over the past few decades, the field of bioinformatics has emerged as the solution to this challenge. Bioinformatics is a discipline centred on the development and application of computational, mathematical and statistical methods for the management and analysis of the vast amounts of biological data now regularly produced by modern high throughput techniques. The field can be broadly divided into two main branches of research: first is the creation of tools that can efficiently process, store and manage data as well as provide intuitive ways to access this information; second is the application of existing knowledge from fields of computer science and informatics, such as machine learning and pattern recognition methods as well as the development of brand new statistical methods with which to analyse and draw meaningful conclusions from the large volumes of data being generated. The informatics challenges of high throughput approaches can be roughly divided into four broad themes:

Biological sample management High throughput approaches can suffer from what is known as the as a large p (dimension) small n (sample size) problem, resulting in large amounts of uncertainty in results generated when using small numbers of biological samples. To combat this problem, high throughput approaches require very large sample sets. For example, very large sample sets, and sample collections are now very common in many research areas, such as genome wide association studies (Zeggini *et al.*, 2008). Therefore the management of biological sample

collections is a very important task, requiring consistent tracking of all manipulations carried out and the accurate recording of sample details including phenotype data. On a large scale, this can only be effectively carried out using a computerised system such as Laboratory Information Management System (LIMS).

Experimental platform management In parallel to the management of biological samples, are the requirements of managing the measured features of the high throughput platform, such as the spotted sequences on a DNA array. It is essential to record detailed annotation on the thousands of features that comprise the platforms of many high throughput approaches, as experimental results are rendered meaningless if these data can not be accurately recalled.

The data deluge The number of data points generated by high throughput experiments is massive when compared with the number of data points generated by low throughput experiments. This increase in the rate of data generation means that researchers often have to deal with result sets that contain millions of observations. The implementation of systems capable of dealing with such large datasets is essential to successfully harness the power of high throughput technologies. For example, relational databases for efficient storage and retrieval of the data and automated pipelines for data processing and quality control.

Analysis of high throughput approaches Along with the need for effective data management strategies is the requirement for new statistical methods capable of analysing data sets which are formed of very large numbers of samples and variables. An example of a successful response to the requirement of methods for the analysis of new high throughput data is the Bioconductor project (Gentleman *et al.*, 2004), an open source and open development software project providing tools for the analysis

and comprehension of genomic data. Hand in hand with the requirement for new statistical analysis methods, the computational hardware requirements which high throughput approaches demand are constantly increasing. These demands include increased processing power and memory for analysis and greater amounts of physical disk space to store all the data constantly being generated.

The field of bioinformatics has done a great deal to manage the challenges created by high throughput approaches; however it is still common for the informatics requirements of high throughput projects to be overlooked.

1.3 Microarrays

An example of a very successful high throughput technology is microarrays. Microarrays have become a ubiquitous technique in scientific research following their first description over fifteen years ago. Microarrays are comprised of a solid support such as glass, on to which specific DNA or protein fragments are immobilised at precise locations. The high accuracy of reporter deposition combined with the tiny volumes applied, both of which are achieved through the use of robotics, have made it possible to manufacture microarrays possessing hundreds of thousands of spotted reporters on a single slide. When combined with differentially labelled samples and the highly specific complimentary binding of DNA, microarrays can be used to survey the transcriptional profile of a cell or detect the levels or DNA copy number in a tumour sample. Microarrays can also be coupled with immunological precipitation techniques to characterise protein binding sites or used to detect nucleotide modifications in order determine the methylation state of a DNA sample.

1.3.1 Array comparative genomic hybridisation

Array comparative genomic hybridisation (array CGH or aCGH) (Solinas-Toldo *et al.*, 1997) is a good example of a very powerful and widely adopted application of microarray technology. The high throughput technique of aCGH is based on a much lower throughput approach called comparative genomic hybridisation (CGH), both these techniques allow researchers to detect and locate genetic alterations as copy number alterations present in the DNA of a sample.

1.3.2 Background

A CGH experiment involves the competitive hybridisation of a mixture of reference and test samples that have been differentially labelled using fluorescent dyes to immobilised whole metaphase chromosomes which serve as the genomic template for hybridisation (Kallioniemi *et al.*, 1992). The fluorescence of the samples across the chromosomes is then measured with the resulting intensity ratio of the reference and test samples being used to determine the copy number of the DNA in the test sample. The copy number of a region of DNA can be determined on the assumption that the reference sample contains a normal diploid genome, such that an increase in the ratio of test sample fluorescence corresponds to an increase in amount of DNA in the test sample which we interpret as copy number gain and a decreased ratio of test sample fluorescence corresponds to a decrease in the amount of test sample DNA which we interpret as a copy number loss. As a technique, CGH has been very successful when applied to the detection of copy number aberrations for many different applications. However, its main limitation is its relatively low resolution of about 5-10Mb (Carter, 2007), meaning that it is not possible to detect smaller aberrations. Also, the use of whole metaphase chromosomes as the genomic template

means that aberrations can not be accurately localised.

Microarray based CGH (aCGH) (Pinkel *et al.*, 1998) overcomes the limitations of traditional CGH by combining CGH with microarray technology, creating a high resolution method for the identification of copy number changes throughout the genome. There are three array based platforms that can be used for CGH; Single Nucleotide Polymorphism (SNP) arrays (Rauch *et al.*, 2004), oligonucleotide arrays (Carvalho *et al.*, 2004) and large construct arrays (Pinkel *et al.*, 1998). The resolution of the imbalances capable of being detected depends on the specific platform used, but which ever platform is chosen, aCGH can detect far smaller copy number imbalances than the traditional CGH method. The procedure for aCGH is the same as traditional CGH in essence, however instead of using whole metaphase chromosomes as the template, aCGH uses a DNA microarray (figure 1.1).

Bacterial Artificial Chromosomes (BACs) offer the best characteristics in terms of genome coverage and background noise levels of all sequences used to produce DNA microarrays (Carter, 2007), originally generated for the human genome sequencing project (Bentley *et al.*, 2001; Consortium *et al.*, 2001). Depending on the genomic depth of coverage of the BAC clone set used BAC arrays are capable of providing resolution an order of magnitude higher than traditional CGH (Theisen, 2008). For example a BAC clone set providing 1.5-fold coverage of the human genome, would give an approximate resolution of 80 kb as this approximately two-thirds of an average BAC clone. The overlapping tiling path design also provides redundancy of markers and ensures there are no gaps between markers meaning fewer alterations will be missed.

In aCGH, and to a lesser extent CGH, researchers have powerful techniques capable of accurately detecting and mapping genetic alterations. The ability of these techniques to detect copy number imbalances between two samples has seen

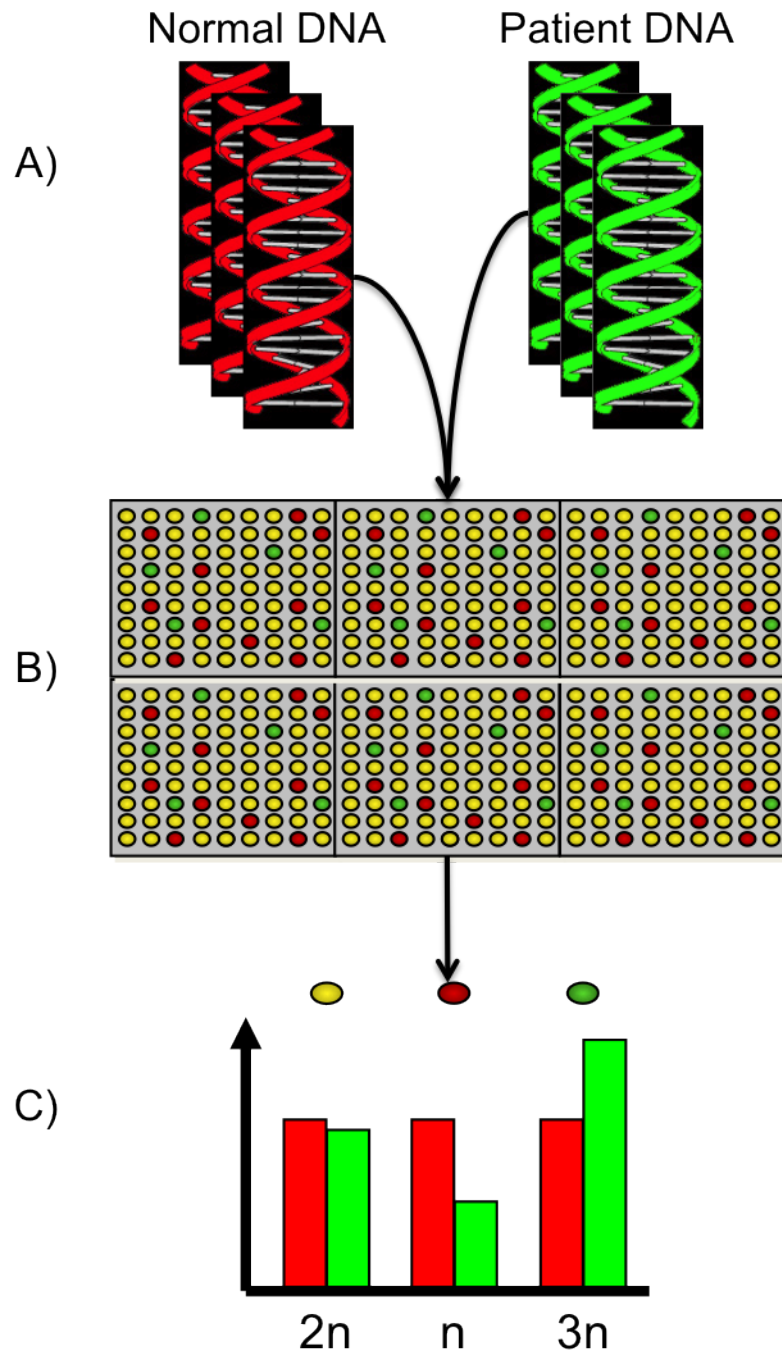


Figure 1.1: This figure shows a schematic representation of the microarray CGH experiment workflow. A) Patient and normal reference samples are differentially labelled using fluorescent dyes. B) The labelled samples are then cohybridized onto a glass microscope slide on which DNA fragments have been immobilized. C) The resulting ratio of fluorescence intensity between hybridised patient and normal DNA for each array spot is measured using a laser scanner and extracted using image analysis software. In this case spots where the amount of patient and reference DNA is the same will appear yellow, where there is a loss of DNA in the patient spots will appear Green as there is more reference DNA bound and spots showing a gain of DNA in the patient sample will appear Green.

them become very popular and widely used in the areas of disease research where genetic abnormalities are involved in the aetiology of the disease, such as congenital disorders and cancers (Lockwood *et al.*, 2006). There has also been work carried out using aCGH to survey the copy number variation that can be found in normal genomes (Redon *et al.*, 2006),(Conrad *et al.*, 2009).

1.3.3 aCGH informatics

Due to their complexity and scale, microarray experiments require a number of computational preprocessing steps in order to transform the data into a form that can be used for further study, namely the copy number of genomic segments. Required preprocessing steps include quality assessment, normalisation of the raw values to remove systemic biases and feature annotation. Other important requirements are the extraction of the raw data and storage of the data.

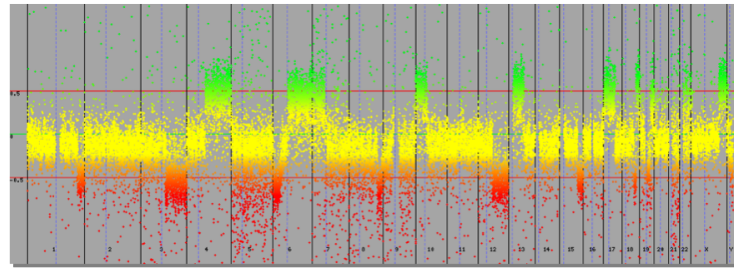
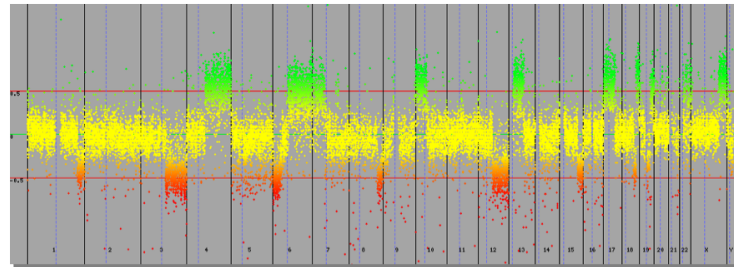
Image analysis and quality control There are various commercial and non-commercial algorithms available to perform the first informatics step of microarray experiment analysis which is known as image analysis. The goal of image analysis is to extract, for each spotted feature, a measure of abundance in the two labelled samples as well as to obtain quality measures (Yang *et al.*, 2001). The image analysis process is a crucial step in the microarray experiment workflow because all the downstream data interpretation is based upon the values created at this stage. Therefore errors introduced at this stage may bias the final data. For a typical two-channel experiment, image analysis involves the processing of the 16-bit tagged image file format (TIFF) files created by the laser scanning of the hybridised microarray slides. Typically the scanned images are produced in pairs corresponding to each of the fluorescent dyes used to label the DNA samples. The processing procedure converts

the raw images into relative intensity values for each spot present on the array. Image analysis is made up of 4 steps which are gridding, segmentation, extraction and quality control.

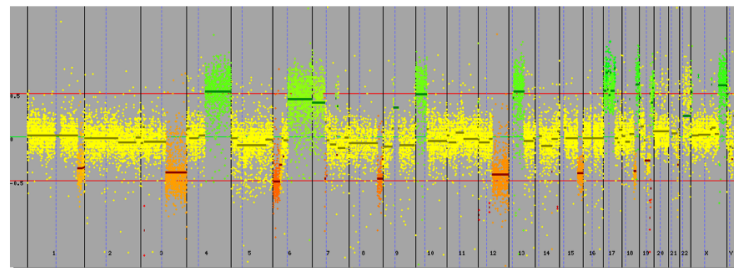
The first step, ‘gridding’, involves the correct mapping of the spotted features to the spots in the scanned image using details of the array design layout typically in ‘GenePix Array List’ (GAL) file format, this step can be performed in a manual, semi automated or fully automated manner depending on the software selected. The next step of ‘segmentation’ is separation of each spot into the background and foreground of the feature signal as the scanning process can produce noise surrounding the spotted features. There are a number of different methodological approaches available for the segmentation of the microarray image, but all aim to separate the biologically relevant signal of the spot from the unimportant background signal or noise. The intensity ‘extraction’ step uses the segmented spots to calculate intensity values for the foreground signal of each fluorescence channel for every spotted feature on the array; this is also the stage at which spot quality scores such as signal to noise ratio, or spot morphology, measures are commonly calculated. The final step of image analysis is quality control, filtering out poor quality spots based on a spot quality score is a very important step for removing poor quality data which could lead to bias in the downstream data analysis. If replicate spots are present on the array, an additional quality control (QC) step of replicate variation can be performed before their signals are combined. Further to spot level quality control, it is possible to identify entire experiments which are producing sub optimal intensity values by visualising the two microarray images overlaid to create false colour microarray images. These images enable the rapid identification of patterns of nonuniform hybridisation and thus the identification of poor quality experiments which should to be repeated.

Normalisation Normalisation is the removal of systemic bias in the data so as to leave only biologically relevant signals. This is an essential pre-processing step for microarray data as noise can be introduced into microarray data at any one of the many complex steps which go together to make up a microarray experiment. The noise introduced can be caused by a number of reasons including unequal quantities of starting DNA, differences in labelling, or detection efficiencies of fluorescent dyes (Quackenbush, 2002). Normalisation is also an essential step as it allows the direct comparison of experiments, as without normalisation this would not be possible due to experimental variation. The vast majority of normalisation methods which have been described have been developed for use with expression array data addressing the factors which affect them. Neuvial *et al* (Neuvial *et al.*, 2006) reported that spatial effects had a greater effect on aCGH data compared with expression array normalisation methods. They went on to describe two types of spatial effect which commonly affect CGH arrays and methods to remove them. The first is a local spatial bias, which is a cluster of spots that have a discrete signal shift unrelated to spotting effects. The second is a continuous spatial gradient which is a smooth gradient in signal from one side of the slide to the other. Methods to correct for these effects have been implemented as an R package called MANOR (Neuvial *et al.*, 2006) as part of the Bioconductor project. Staaf *et al* (Staaf *et al.*, 2007) have also implemented an aCGH normalisation package in R in which they take copy number imbalances into account during normalisation as they that copy number imbalances correlate with intensity in array-CGH data thereby causing problems for conventional normalisation methods.

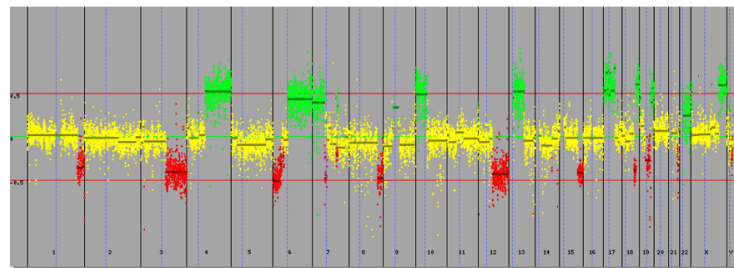
Segmentation The penultimate step of pre-processing an experiment is the identification of locations with copy number transitions or breakpoints, so that

(a) Raw \log_2 intensity data

(b) Normalised and filtered data



(c) Segmented data



(d) Copy number called

Figure 1.2: Microarray CGH informatics pipeline steps illustrated using aCGH plots of experimental data following each step of the pipeline. The aCGH plots show each feature on the microarray plotted according to its \log_2 intensity ratio on the y axis and its genomic location of the x axis. a) The raw unnormalised, unfiltered \log_2 intensity ratios show the high level of noise in the data which needs to be removed to leave only the biological variation under study. b) After normalisation and quality based filtering procedures the noise in the data is reduced. c) Following segmentation the identified regions of similar copy number are show using the horizontal bars plotted at the average \log_2 intensity for the segment. d) Finally following copy number calling and window based smoothing the segmented regions are called as either normal (yellow), gain (green) or loss (red) and appear as much tighter clusters of points.

the genome is divided into regions of equal DNA copy number. The segmentation step enables us to simplify the data and calculate the average log ratio for each defined segment of continuous copy number. Segmentation is one of the most extensively studied aspects of the aCGH analysis process and thus there have been a large number of methods suggested to meet this challenge using a number of different statistical techniques, including intensity thresholds (Hodgson *et al.*, 2001; Vermeesch *et al.*, 2005), wavelets (Hsu *et al.*, 2005; Wang and Wang, 2007; Huang *et al.*, 2008), circular binary segmentation (Venkatraman and Olshen, 2007), gaussian model-based approaches (Hupé *et al.*, 2004), quantile Smoothing (Eilers and de Menezes, 2005), hierarchical clustering-style trees (Wang *et al.*, 2005), penalized likelihood criteria (Picard *et al.*, 2005), expectation-maximization-based methods (Myers *et al.*, 2004), genetic local search algorithms (Jong *et al.*, 2004) and hidden Markov models (Fridlyand, 2004; Marioni *et al.*, 2006; Shah *et al.*, 2006).

Calling The final step of processing an aCGH experiment is ‘copy number status calling’, which involves normalised log-ratios being converted back to an absolute measure of one of four underlying discrete states representing loss (<2 copies), normal (2 copies), gain (3–4 copies), and amplification (>4 copies) (van de Wiel *et al.*, 2007). Using the process of calling, the results of an aCGH experiment can be expressed in a far more biologically relevant way using copy number states, as opposed to continuous log-ratio values. The process also has the advantage of typically reducing the number of copy number segments when compared with the number of segments produced through segmentation, making reporting of results, and any downstream analysis, simpler.

The results of the aCGH informatics steps can clearly be visualised in figures

1.2(a) to 1.2(d) where the effect each processing step has on the data can be seen. The transition from noisy raw data in figure 1.2(a) to the far more discrete processed data in figure 1.2(d) demonstrated the value and importance of these informatics steps in the aCGH experimental procedure.

1.3.4 Downstream analysis

The steps described in the aCGH informatics section 1.3.3 comprise the procedures required to analyse a single sample. Once this process is complete, a map of the regions of genomic gain and loss for the sample will have been generated. With such a map of genetic aberration, much can be learnt about the genetic status of a sample, but in order fully to realise the capability of aCGH this process needs to be repeated across multiple samples in a high throughput fashion. Downstream analysis is the term applied to any analytical steps performed on the processed aCGH results, typically involving multiple samples. There are a number of types of downstream analysis that can be applied to aCGH data:

- Clustering of samples for subtype discovery, is a form of downstream aCGH analysis performed using unsupervised machine learning algorithms. Machine learning is the development of computer algorithms for performing desired tasks using using example data or past experience (Larrañaga *et al.*, 2006). Unsupervised learning techniques do not require any initial data to learn properties of the data, they instead attempt to find structure and patterns within a given data set.

Cluster analysis is an unsupervised machine learning approach that aims to identify groups within the data that are similar to each other, this type of analysis is performed on the assumption that the groups share some interesting

functional similarity. This form of analysis has already been used successfully in the detection of distinct type of renal cell carcinomas for diagnostic purposes (Wilhelm *et al.*, 2002) and to show that chromosomal aberrations of tumours from hematopoietic and mesenchymal origin are distinct to tumours of epithelial origin (Jong *et al.*, 2007). The detection of molecular subtypes of cancer, that clustering analysis allows could be very important for improving diagnosis and treatment of disease.

As aCGH data presents a unique challenge for clustering in the discrete nature of called data there are only a few published methods for performing clustering analysis with aCGH data (van de Wiel *et al.*, 2011). WECCA (weighted clustering of called aCGH data) (Van Wieringen *et al.*, 2007) is a hierarchical clustering method. WECCA produces clusters using a stepwise process: first weights are assigned to each clone or region and then the similarity between all cluster pairs is calculated and the two clusters with the highest similarity are merged. This process is repeated until one cluster remains. The hierarchical tree formed by this process can then be partitioned in separate groups by sectioning the tree at a chosen cut-off level, with the cut-off level chosen such to produce compact and well-separated clusters. Another distance based method that works in a similar way to the WECCA method has also been published, it however uses k-means clustering rather than hierarchical clustering (Liu *et al.*, 2006). The final clustering method available for aCGH data is a HMM model based approach (Shah *et al.*, 2009)

- Identification of common regions of copy number loss and gain across multiple samples provides researchers with an effective way of identifying the most frequent, and thus more important, copy number alterations. Recurrent

alterations are most likely to be driving the given phenotype under study. The simplest way to achieve this is using frequency thresholds, but this approach is slow, prone to investigator bias and is unable to identify aberrations within subsets of samples. Computer algorithms which are able to perform this type of analysis automatically include, STAC/MSA (Diskin *et al.*, 2006; Guttman *et al.*, 2007) and KCSmart (Klijn *et al.*, 2008) which all use a permutation based approach, Hierarchical HMM (H-HMM) (Shah *et al.*, 2007) which uses a hidden Markov model and MAR/CMAR (Rouveirol *et al.*, 2006) which does not use a statistical model.

STAC and its subsequently improved upon method MSA both use permutation of regions within chromosomes to assess significance. STAC and MSA are both available as Java applications, however they differ in the input they require and in the output they generate. STAC requires input data to be segmented, MSA only requires normalised log ratio values. STAC generates confidence values for the identified recurrent regions as output where as, MSA returns a p-value for each region. The KCSmart algorithm calculates an average ratio intensity across samples for each array probe to identify recurrent regions, combining positive and negative ratios separately. A Gaussian locally weighted regression is then applied to the summed totals. The results of this process is then corrected for the non-uniform distribution of probes along the genome that occurs on aCGH platforms. The identified peaks are then tested using a permutation approach against a randomly permuted background. Significant recurrent regions of copy number gains and losses are finally determined using the resulting, multiple testing corrected, significance threshold. The KCSmart algorithm has been implemented in R and is available as part of the Bioconductor project. The R implementation requires normalised positive and

negative intensity ratio values separately as input.

The MAR and CMAR algorithms find minimal common regions using segmented data. These two algorithms are not suitable for the development of downstream analysis pipeline as neither produce a test statistic that can be used to assess the confidence for a minimal common region. The source code for the both algorithms is also not publicly available, where the source code for published methods is only available upon request of the authors as is the case for MAR and CMAR is a very bad situation. “If the source code is good enough to do the job, then it is good enough to release and releasing it will help your research and your field” (Barnes, 2010).

The H-HMM model extends a previously developed single sample aCGH HMMs (Shah *et al.*, 2006) to deal with multiple samples to detect shared copy number aberrations. The algorithm uses raw data for input to avoid filtering out important signals in the raw data. The algorithm has some clear limitations, the probability shared copy number aberrations is produced for each array marker not for regions, the algorithm can not identify subgroups within the data and unfortunately the code for running the H-HMM is only available for MATLAB (MATLAB, 2012), making this method inaccessible to researchers that do not have a MATLAB license.

There has been no comprehensive comparison of the different approaches, and very few of the published papers present any comparison with other methods. The field is therefore ready for a comprehensive, careful, comparison of the relative strengths of available methods using a variety of simulated data sets to better understand which methods will perform better on different real data sets (Rueda and Diaz-Uriarte, 2010).

- Classification by genomic profile for prognosis and diagnosis is a form of downstream aCGH analysis most commonly performed using supervised machine learning algorithms. Supervised machine learning algorithms use an initial set of data known as a ‘training set’ to infer information about the properties of the data. The inferred information can then be used to make predictions about other data. Although there are a number of different types of machine learning algorithms such as nearest neighbour (Dasarathy, 1990) or bayesian classifiers (Duda and Hart, 1973) the best performing algorithms for aCGH data are Support Vector Machines (SVMs) (Wang *et al.*, 2006).

Support vector machines (SVMs) (Vapnik, 1995), are a computational technique for classifying high dimensional data such as that produced by microarrays (Li *et al.*, 2004). SVMs are supervised machine learning algorithms for binary classification. Supervised machine learning algorithms use an initial set of data known as a ‘training set’ to infer information about the properties of the data. The inferred information can then be used to make predictions about other data.

SVM classifiers are constructed by first identifying the optimal hyperplane that maximises the margin between all the support vectors (shown in figure 1.3), this hyperplane is then used to classify subsequent new data. The larger the margin, the better the generalisation of the classifier thus avoiding overfitting. Overfitting is a phenomenon caused by machine learning algorithms too closely generating a classifier that is too well fitted to the training data and thus poorly generalisable causing it to perform badly on new data.

The random forest (Breiman, 2001) is becoming a popular technique for classification of aCGH data, it has developed an excellent reputation amongst

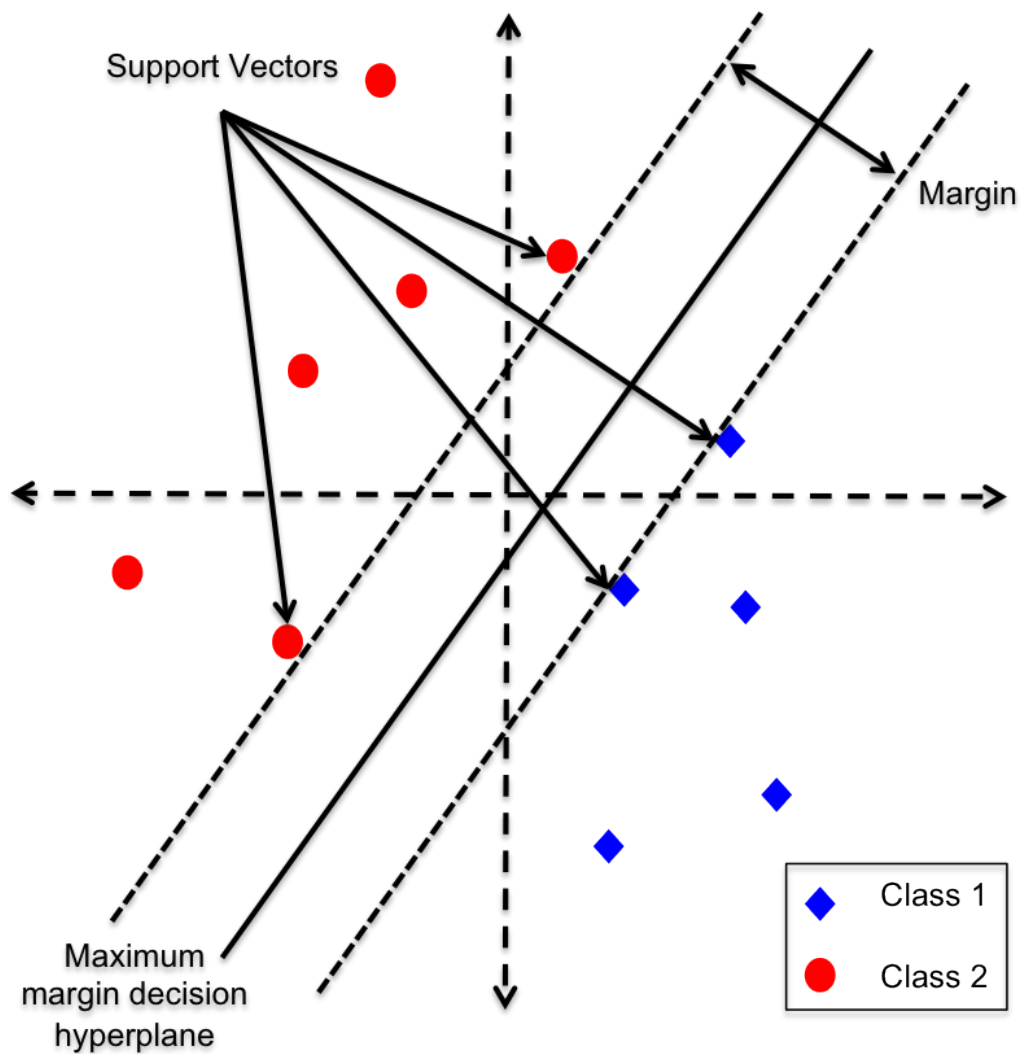


Figure 1.3: This figure shows a schematic representation of a simple support vector machine classifier for two classes of data. The two blue and two red points that constrain the width of the margin are the support vectors. The support vector machine constructs a classifier by finding the hyperplane that maximises the margin between all the support vectors.

the statistics and machine learning communities as a versatile method that produces accurate classifiers for many types of data (Amaratunga *et al.*, 2008). In a random forest, a tree, is trained on a sample of cases drawn at random from all cases and each tree uses a random subset of all the features to build its classifier. The forest is an ensemble of some number of such trees, where each tree is called a base classifier. Classes are assigned to samples by majority vote: when given a test case, each tree assigns it a class according to its classifier; this information is collated and overall the forest assigns it the most frequent class. The out-of-bag cases in any tree can be regarded as test cases for that tree as they were not used to build it and thus they can be used to assess the performance of the forest as a whole; this is done via the out-of-bag error rate, which is the proportion of times an out-of-bag case is misclassified. A major advantage of random forests is that they are able to keep the likelihood of overfitting low by using different subsets of the training data and different subsets of features for training the different base classifiers. This means that only patterns truly present in the data would be detected consistently by a majority of the base classifiers and the majority votes turn out to be good indicators of class (Amaratunga *et al.*, 2008).

- Combining aCGH data with the results of other high throughput genetic techniques is a common downstream approach attempted by a number of groups, researchers have most frequently attempt to correlate copy number loss and gain with gene expression levels (Chin *et al.*, 2006; Lapointe *et al.*, 2007; The Cancer Genome Atlas Research Network, 2008; Haverty *et al.*, 2009; Savola *et al.*, 2009).
- Identification of likely causal genes in identified regions of copy number

aberration is another very useful downstream analysis approach. There are a number of ways in which these genes can be investigated, for example by looking for gene ontology terms that are over or under represented, researchers can also look at the pathways in which any identified genes lie to give clues to the biological implications of the copy number alteration. As is very often the case in high throughput research, a very large list of genes may be generated using this approach and, in this case, it may be useful to apply an unbiased prioritisation procedure to reduce the number of potentially interesting genes (Furney *et al.*, 2008).

1.3.5 Laboratory information management systems

As is illustrated in section 1.3, high throughput techniques such as microarrays involve numerous experimental and informatics steps in a single experiment, with each step capable of generating vast amounts of heterogeneous data. The accurate management of all of these data is essential in ensuring that high quality reproducible results are consistently generated.

By far the best solution to the massive data management challenge posed by high throughput technologies is the implementation of a Laboratory Information Management System (LIMS). A LIMS is an integrated informatics solution designed computationally to capture the workflow of an experiment accurately and consistently by recording details of every step including all the associated data; this task is completed automatically where possible. LIMS also need to provide intuitive interfaces for users for data retrieval and visualisation. A suitable LIMS solution capable of managing projects which employ the technique of microarray CGH would require all the following features:

- **Sample management** The accurate recording of all the details of every sample used in a project is a very important function performed by LIMS. The recording and retrieval of detailed sample annotation information including phenotype data is key to extracting maximum information from an experiment.
- **Recording experimental procedures** Details of every laboratory procedure performed on a sample need to be recorded to ensure the design of fair experiments where all samples have undergone the same pre-processes so that only the biological phenomenon of interest is under study.
- **Interaction with laboratory instruments** All high throughput techniques rely upon sophisticated laboratory equipment including robotics for sample processing and laser scanners for data capture. Therefore it is necessary to be able to communicate efficiently with these essential pieces of equipment either to provide input data to direct the equipment or to extract resulting data from the equipment.
- **Pre-processing and analysis** The ability to perform basic data pre-processing quality control and analysis steps, such as data inter- and intra-experiment normalisation.
- **Data visualisation** Visualisation of data is an essential feature of a LIMS as it allows users very quickly and easily to assess an experiment. Microarray LIMS typically include the ability to view both the raw and processed versions of the experimental data, together with different quality control plots that allow users to assess their experiments quickly.
- **Tracking of batches** The ability to track batches of experimental reagents, or microarray slides, is an important feature that also highlights one of the

big advantages of using a LIMS. Because all experimental data are collected, recorded and integrated, it is far easier and quicker to track down and diagnose problems with batches as they arise rather than after a project has been completed, especially when batch information is incorporated into aspects of data visualisation.

- **User management** Owing to their scale, high throughput projects can typically involve a large number of researchers, each of whom might have different roles within the project; thus the ability to control what data each researcher has access to is a very useful tool which can be controlled using user accounts.

LIMS solutions

A LIMS can be deployed in a laboratory using one of a few different approaches: firstly a closed source commercial package can be purchased; secondly a free open source application can be obtained; or finally a bespoke solution can be developed in-house. There are clear advantages and disadvantages to each of these approaches. Commercial applications are ideal in highly time sensitive scenarios as these solutions should be quick and easy to set up as they are designed to ‘work out of the box’. However disadvantages of commercial LIMS include the cost to purchase the required number of licenses and the correct level of support. The closed source nature of commercial LIMS also limits the ability to extend and enhance the software to meet specific needs. While new features in commercial LIMS can be requested by users, final decisions and the time scale of their deployment are at the behest of the developer. Open source solutions such as the highly successful BASE (BioArray Software Environment) platform (Saal *et al.*, 2002; Christersson *et al.*, 2009) are free

to obtain; however they do typically demand more time and knowledge to setup. Extending and improving open source solutions is also far easier than commercial applications; however if significant areas of functionality are missing, a bespoke LIMS could prove to be the best solution.

The level of activity surrounding an open source software project is another important consideration as bioinformatics projects can often stagnate due to the high rate of personnel turnover in academia, ultimately resulting in an unsupported application. The development of a bespoke LIMS can lead to the deployment of a solution that meets all the requirements of a laboratory and also ensures the quality of the software which impacts on the reliability and reproducibility of data in the system. While the choice to develop a bespoke LIMS application is the most time consuming option of the three possible LIMS solutions, development time can be dramatically improved with the application of available programming libraries and frameworks (Morris *et al.*, 2008a).

1.4 Software development for high throughput science

Scientific research using high throughput techniques is an approach that has thus far proved to be highly successful in the discovery of novel scientific findings and because of this they will remain the primary tool for making new discoveries for some time. In order to fulfil this very high promise, high throughput technologies still require further improvement of their informatics support which is so vitally important to manage the unique demands of these approaches.

In summary, high throughput techniques such as microarrays have some very challenging informatics requirements that can only be solved through the development of high quality solutions. The first and most obvious challenge of high

throughput approaches is the volume of data generated. Software solutions for high throughput techniques should incorporate the ability to communicate with relational databases, especially when data from different approaches needs to be interrogated together. The next challenge posed by high throughput techniques is the rapidly changing data format and analysis demands. One possible solution to the difficulties that changing demands in data processing and analysis present is the development of more modular flexible pipelines which are capable of very easy customisation and the ability to plug in new data processing elements and analysis methods. The final challenge that is clear from the previous two sections is the requirement of software which generates results that are both accurate and highly reproducible, a challenge that can only be met through the development of successful software projects of high quality.

Bioinformatics projects, much like software projects in many other areas fail and they fail all too often. This is an unfortunate fact that developers have been aware of for a long time. To understand why so many projects fail we must first begin with a definition of how we measure a successful software project. A successful project should deliver a piece of software that is ready to use on time; the cost of the project to produce the software should be what was expected; the software serves the purpose for which it was designed; the software must also not be crippled by bugs; and the final measurement for a successful project is that the software produced should be used to make a positive impact on the area for which it was created (Bain, 2008).

1.4.1 Software development processes

In order to bring about a change in the proportion of failing software projects, software developers began to think about ways to improve the way software was

developed. The idea agreed upon by most was somehow to formalise the process that was required to create high quality software. The thinking at the time was that to prevent failure of a project, software development needed to be managed using a process that formalises the steps that are required to deliver a successful piece of software in a timely fashion. Owing to the fact that the field of software development was a fledgling one when these issues were first being discussed, people decided to adopt methods from other more established fields such as engineering. One of the first software development process put forward that is still in limited use today is called the ‘waterfall process’.

The waterfall process used engineering principles for the development of software; the process is comprised of a number of required phases of work which are completed in sequence, so as one phase finishes the next phase down begins, such that work in the project flows down through each phase like a waterfall. The phases or steps typically involved are the analysis phase where detailed analysis of the problems which needs to be addressed is carried out; the design phase where the design of a suitable solution for the given problem is formulated taking into account any given constraints; the construction phase is where the design is implemented and is the first place where any source code gets written; and the final phase is testing to ensure the software meets the requirements identified in the initial analysis phase.

However, the use of the waterfall process for software development very often produced software that did not meet the expectations of the customer and projects using this process tend to take more time and cost more money than was originally planned. One of the major faults of the waterfall process is that it places far too large an emphasis on the analysis and design phases which are very time consuming as they involve the production of large amounts of documentation. Another failing of this process is that after the design phase is complete, there is no built in way of

changing the design, thus any changes would have to wait until after construction and testing is complete, this is why so often software was created which no longer met the requirements of the customer.

The rigidity and thoroughness of the waterfall process for project management is what originally made this approach very successful for large engineering projects (Bain, 2008) but at the same time is the reason why using this approach for software development so often ends in failure. The most common cause of the failure of software development projects is change; either change in the functionality required, a change in the problem being solved, or a change in the demands of the customer. The waterfall process contains no way to allow for inevitable changes that will occur during a software development project which is why this process (although still used for a small number of projects today) is regarded as unsuitable for the development of successful software.

Variations of the waterfall process have been suggested that have since tried to address the flaws of the original process by increasing the use of feedback, such that feedback from the testing phase leads to further design to correct elements of the design that do not work. Feedback from design should lead to revision of the requirements analysis where requirements are identified as impossible to design. Some processes have taken this idea a step further by involving feedback at every stage, these modified processes have improved the original waterfall process, but they do not solve all the problems. Because software development is a unique problem that requires its own approach and not simply a variation of an engineering solution.

Agile

An alternative approach to these so called ‘heavyweight’ processes are ‘lightweight’ flexible and dynamic processes that are better suited the nature of software

development. There are a set of methodologies which fit this criteria and that have also been gathering a large amount of interest in the field of software development. These methodologies are part of an exciting shift in software development called the ‘agile movement’; the agile movement manifesto was created in 2001 (Cunningham, 2001), and has four core values:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The manifesto states that while there is value in the items on the right, the items on the left should be valued more. Agile development processes provide more flexible alternatives to the traditional ‘heavyweight’ methodologies, and because change is one of the most common reasons for the failure of a software development project, this in turn means that agile processes should be far more successful in delivering successful software. There are a number of different agile methodologies that each interpret the agile manifesto slightly differently; these include dynamic systems development method (DSDM) (Consortium, 2012); crystal methods (Cockburn, 2004); feature-driven development (Palmer and Felsing, 2002); lean development (Poppendieck and Poppendieck, 2003); adaptive software development (Highsmith, 2000); scrum (Schwaber and Beedle, 2001); and extreme programming (XP) (Beck, 1999). The two most popular and widely used of these methodologies are scrum and XP which are described below.

Extreme programming

Extreme programming turns conventional software development processes sideways by planning, analysing and designing a little at a time throughout the project rather than performing these tasks in discrete phases. The methodology is based on twelve practices or guidelines which are intended to be used together (Beck, 1999).

Planning game The first practice termed the ‘planning game’ concerns the initial analysis of the problem and the estimation of the time it will take to create a release. In XP ‘stories’ are used to define what the software should be able to do; a ‘story’ is simply a description of a use case for the software that fits onto a small piece of paper, and which is testable and estimable. Once analysis has finished and all the stories have been created, a release estimate can be generated. This can be approached in different ways; the customer can either choose the stories they want included in the first release, with the developers then providing an estimate of the time those stories will take to complete, or the customer can provide the developers a time for the first release allowing the developers then to specify which stories can be completed in the given time period.

Small releases Software releases in an XP process are quick and often. The first release will typically be before all of the stories have been completed and, after that, releases should follow regularly. With each release, the customer should define the next most important story to implement. This practice is opposite to ‘heavyweight’ processes which only release code when all the software specifications are fully met and so allows for a great deal of flexibility and rapid change and provides a sense of accomplishment that is often missing in long projects.

Metaphor The shape of the system is defined by a metaphor, or set of metaphors, shared between the customer and developers. The metaphor provides a broad aim for an XP project.

Simple design XP puts emphasis on the creation of simple solutions that achieve the given task. In this way new code can be released quickly and, if the solution is too slow, this can be addressed later. It is very often the case that the simplest solution is the best and time spent optimising code may be wasted.

Tests Tests are one of the most important practices in XP, used differently from in ‘heavyweight’ methods where testing is performed after the software is constructed as part of quality control, by someone other than the person who wrote the code. Tests in XP are performed by the developer before the software is written using what is described as the ‘test first’ technique. The test first technique means that no production code should be written before a test exists for the feature being developed. When the test is first created, it should fail. Development then takes place until the test passes, at which point the feature can be considered to be implemented and working correctly. (If it were to pass, that would indicate that the feature already exists, and therefore no coding would be required to implement the feature.) Testing has a number of features which make it suitable as a very useful development tool and not just for quality control. Testing creates confidence in software, and once written, tests can be run multiple times with no additional effort. The use of tests in development means bugs can be identified much more quickly and it can also highlight potential design problems. For instance, if a solution is very difficult to design a test for then it may not be the simplest solution.

Refactoring Refactoring is a practice that involves improvement of the internal quality of source code without changing the external behaviour of the software. This is almost impossible to carry out without a fully developed test suite, as the test suite forms a safety net which ensures that any refactoring has not broken the working code. Refactoring may at first seem a pointless practice as it does not result in additional features. However, it makes code easier and quicker to maintain and simpler to extend in the future, since with each round of refactoring, the code quality should improve.

Pair programming The practice of pair programming is definitely a case where the extreme in ‘extreme programming’ comes into play. Using the pair programming practice, all code is written by two developers working at a single computer with a single keyboard and mouse. The developers should switch between who is ‘driving’ the coding, either regularly or when one programmer reaches a block in progress. The switching of partners is also recommended to prevent the practice becoming stale. As with the practice of refactoring code, pair programming might at first seem a waste of resources with one developer in the pair not producing any code. However, like refactoring, the benefits of pair programming come from improvements in the quality of the code produced. Far fewer bugs are introduced as a result of the constant peer review of the code and what is created is normally far more interpretable code as it must make sense to both programmers at the time of production. The improved communication within the development team that pair programming creates is another good reason why this practice is beneficial.

Continuous integration Any new code should be immediately integrated with the current system as soon as it is completed and has passed the necessary tests instead

of waiting for release dates.

Collective ownership Any programmer should endeavour to improve any code anywhere in the system if the opportunity arises.

On-site customer For every project a customer, that is someone who is going to use the software being developed, should work in the same office as the developers. This allows very rapid feedback for queries relating to the project, which produces software much closer to the customer's specification.

40 hour weeks Developers working significant amounts of overtime is often a cue that there is something wrong with how the project is progressing, so the practice of 40 hour weeks states that no one can work a second consecutive week of overtime, rather the difficulties with the project should be addressed instead.

Open workspace Having an open space office is another practice relating to the working environment much like the on-site customer practice. This practice states that the development team should all work in the same office space, as this makes practices such as pair programming much easier and enables better communication within the development team for things such as the exchange of ideas which benefits the progress of the project.

Just rules Once a development team has chosen to follow the rules of XP it does not mean they can not go back on them, however they should ensure that they agree on a new set of rules for the project.

Scrum

The scrum process developed by Ken Schwaber has at its core the belief that software development is not a defined process but an empirical process that requires constant monitoring and adaptation. Schwaber believes you cannot predict or defiantly plan what a software project will produce, but you can control the process with monitoring and constant feedback (Highsmith, 2002). Unlike XP, Scrum instead concentrates on the project management aspects of software development. The scrum process takes place in three stages: pre-sprint planning; the 30 day sprint (which is the emphasis of the process); and the post-sprint meeting.

Pre-sprint planning The first stage of the scrum process involves two backlogs, where a backlog is a record of features that are similar to ‘stories’ in XP. The product backlog is all of the features the software requires to complete the project, but to begin a sprint requires the creation of a ‘sprint backlog’ which is the agreed features which are to be completed during the next 30 day sprint. Along with setting the sprint backlog the pre-sprint meeting also needs to decide on a sprint goal; the goal should give a business reason why the chosen features are to be added. These provide a target for the 30 day sprint, it is not a failure if a few features are not completed during a sprint, as long as the business goal is reached the sprint is a success. The sprint goal also helps the team by preventing developers from over-focusing on a particular feature for too long.

30 day sprint Once the sprint backlog has been agreed upon, team members sign up for tasks, this process being guided by the expertise and experience of each team member. One very important feature of the sprint is that feature priorities do not get changed during the sprint. This is because when everything else can change

something has to remain constant. The other important practice during the sprint aims to combat time consuming meetings which so very often accomplish nothing. Daily scrum meetings are the answer to this and they also help to manage the uncertainty and change of software development projects. Scrum meetings are very important to the scrum process and therefore have a tight set of guidelines:

- The meeting should be held at the same time and place every day
- They should last no more than 30 minutes, the ideal being 15 minutes
- The scrum meeting should be attended by all members of the team, managers should attend although they should not participate and only listen
- They should be used to raise problems but not to work on solutions, this is instead carried out by the relevant team members after the meeting
- At each meeting each team member should address three questions
 - What have you done since the last meeting?
 - What are you going to do before the next meeting?
 - Is there anything getting in the way of your work?

Post-sprint meeting After the 30 day sprint has finished a post-sprint meeting is should be held to demonstrate the new features developed during the sprint to the customers, and at the end of the post-sprint meeting the scrum process starts again with the planning for the next sprint.

1.4.2 Software development best practices

Along with following an appropriate development process such as XP or scrum, there are additional practices that can be adopted by bioinformatics software developers

that can greatly increase the quality, productivity, usefulness and maintainability of the software projects under their development.

Source code reuse

Source code reuse is a practice by which software developers utilise and extend existing programming solutions rather than developing a solution from scratch. This practice can very easily improve the quality and productivity of a development project. Software quality can be improved by the incorporation of high quality code from internal or external sources and productivity can be improved as developers are required to spend less time working on common elements of functionality, therefore able to concentrate on solutions to more novel challenges.

However if no effort is made to reuse source code within a project, then redundancy in the code base will increase over time. This is because bioinformatics tasks will often involve common development challenges, such as handling the input and output of files in standard formats. High levels of redundancy in the source code of a project is far from optimal as this has the effect of increasing the number of lines of code unnecessarily. The more code that is written, the greater the opportunity for introduction of errors, a larger code base is also more difficult to maintain and document.

Source code reuse can be achieved in different ways: using libraries of common functions, Application Programming Interfaces (API) to useful resources, modifying code from other open source projects or on a more local scale, the use of a modular development approach such as Object Oriented (OO) programming.

Programming libraries There are a number of repositories of programming libraries available, these containing solutions to a vast amount of common development

problems, such as the comprehensive R (cran.r project.org, 2012) and Perl (Perl.org, 2011) archive networks (CRAN and CPAN) as well as more specific bioinformatics projects for Perl, Java and Python (BioPerl (bioperl.org, 2011), BioJava (biojava.org, 2012) and BioPython (biopython.org, 2012)).

APIs APIs are a set of programming tools which allow developers to communicate with other software using specified functions, data structures and variables. The Ensembl API (ensembl.org, 2012) is an example of a very popular and successful API for the bioinformatics community. This is because it allows the user, in only a few lines of code, to connect to the Ensembl databases, and generate objects that represent entities in the underlying database, which provide multiple retrieval methods (Rios *et al.*, 2010)

Open source programming projects Web-based repositories of open source code such as SourceForge (Geeknet, Inc., 2012) and GitHub (GitHub Inc, 2012) allow users to download useful tools and applications. They also provide the developers of such tools and applications with the ability to collaborate on and share the programming solutions underlying their software. Having the source code of a relevant program available can be very useful for a development project as it can quickly provide tested solutions to what might be difficult tasks.

Object oriented programming Briefly in object oriented programming, software ‘objects provide controlled access to collections of data and a class defines a type of object and describes what data are accessible and how to access it. OO programming has many benefits including: simpler analysis methods, a domain-oriented approach to design cleaner and more compact code, more robust implementations, greater code modularity, easier debugging, more comprehensible interfaces to modules,

better abstraction of software components, less namespace pollution, greater code reusability, scalability of software and better marketability of the final product (Conway, 2000). There are number programming languages that can be used to develop software in an object oriented manner such as Java (ORACLE, 2012), C++ (cplusplus.com, 2012), Perl (Perl.org, 2012), Python (Python Software Foundation, 2012) and R (r-project.org, 2012). All of the languages implement OO programming in slightly different ways, but all result in providing the same advantages to the developer.

The most important and useful OO programming concept in terms of source code reuse is inheritance. Inheritance is a technique that allows the developer to import common functionality quickly into a new class. Typically a ‘base’ class is created containing general variables and functions, then more specialised child classes are able to inherit the general variable and functions from the ‘base’ class, which they can add to with more specialised variables and functions. Inheritance allows developers to quickly reuse the variables and functions they have already created thus reducing development time and reducing errors.

Version control systems

Even modestly sized programming projects can easily generate a very large amount of computer programming source code and associated files. Through the implementation and regular use of a version control system (VCS) such as CVS (Price and Ximbiot, 2006), subversion (The Apache Software Foundation, 2011) or git (Chacon, 2012), a development team or single developer can avoid a number of problems which can arise during the course of a project.

A version control system is software which manages the recording and retrieving of changes within in a project. Although version control systems all differ slightly

in their functionality there is core of functionality that all of these systems possess. One example is the ability to merge files without loss of work where two or more developers are working simultaneously on different parts of the same file, a task which if attempted by developers themselves would certainly be prone to error. Other core features include the ability to retrieve any stored revision of a file for viewing or editing, and the ability to store or retrieve notes concerning a revision. Together, these functions make the process of identifying and fixing bugs in the source code a far simpler task. Additional features typically supported by version control systems include the ability to retrieve difference between versions of files (which makes it easier to create patches or locate bugs) and the branching of projects (allowing the project to progress along multiple development tracks simultaneously). These branches can be merged back into the main line of development when complete.

The workflow of a developer using a version control system differs slightly, but not significantly, from the basic ‘open a file, edit the file and save the file’. To begin development using a version control system first requires making a copy of the repository which is the saved collection of all the files, as work is done on a copy of the repository and not directly on the repository itself. The copy of the repository can be on the same machine as the repository, on a machine on the same local network, or on a machine connected through the Internet. The final connection method means that it is possible for developers in different locations around the world to contribute to the same project; this is very common in the development of a number of large open source development projects. Once the developer has their copy of the repository, they can resume the simple process of opening a file, editing the file to add a feature or fix a bug, and saving the file on any file in the repository. To save the new version of the file in the repository, the developer must first ensure they have the most up to date version of the repository to make

sure another developer has not modified the same file. If the changes made do not conflict with any changes made by another developer, the file may be committed to the repository along with a note; but if a conflict in the file is detected this must be resolved before the changes can be saved. As can be seen in this example, the use of a version control system for a project involves additional work, however the benefits of a well operated versioning system can clearly mitigate the small amounts of developer time and effort required.

Testing

Testing is an essential process in the quality control of software development and is used to ensure software works in the way it was intended. With bioinformatics applications, it is very important to have a level of reassurance about the quality of software, in order that the user can be confident that results generated by the software are correct.

Many developers use a very simple system to accomplish this aim, by compiling the code and then trying features to check they work, or they may create a simple script to test the functions they have implemented. This simple functional testing approach to what is a very important process in software development is far from sufficient to produce successful high quality software. However, it is better than a total lack of testing, an occurrence which I have observed on numerous occasions in the work of others in the bioinformatics field and also through downloading the source code of software published in leading bioinformatics journals.

Testing has other benefits beyond simply testing that a certain function produces the result that is expected. For instance, testing helps during the processes of refactoring and development of new features as they make it very easy to ensure that any changes that have been made do not break the existing working code. Also

tests help to document the code, as they serve as an example of what the code is supposed to do.

There are different types of testing for software development projects such as customer acceptance testing, integration tests and load tests. But by far the most useful form of testing for software development is unit testing. Unit testing is a testing approach where by a test is created for every functional unit in the system. This approach ensures complete coverage of all the code in a project that may not be achieved by simply testing individual features. Unit tests can be written after the production code has been written, or before as proposed by agile methodology. The process of testing and unit testing is well supported by all modern programming languages, for example the JUnit framework in Java that provides classes to extend and an automated test runner that organises the execution of all the tests. There are also numerous modules freely available to assist in writing test in Perl such as Test::Harness, which also automates the running of tests.

Documentation

The production of clear and thorough documentation is essential for the usability of any produced software, but is also very important for the maintainability of the code base and reusability.

It is not only software documentation of bioinformatics software which is important, but also a much overlooked practice is the recording of the steps involved in a bioinformatics experiment using the developed software. It is a well enforced practice in wet laboratory experiments that all the details of an experiment are systematically recorded; this is however by no means normal in bioinformatics. The recording of the analytical steps taken as part of a bioinformatics experiment are just as important as a wet laboratory experiment and, for the same reasons are crucial

for the replication of work. There are a number of simple solutions to manage an electronic laboratory book, wiki-based systems provide a very flexible way to record research and they are also very suitable for collaboration which is very important in science.

1.4.3 Programming languages

The most important software tools that were used in the process of carrying out this project were the programming language Perl, the statistical programming environment R and the relational database management system MySQL. The principles and practices that will be discussed throughout this work are by no means limited to these specific tools. In fact all of the work I will describe could be repeated using a different set of software development tools to achieve a similar goal. There are however very strong reasons why the specific toolset of Perl, R and MySQL was chosen: all of these tools are very widely used solutions both in academia and industry for the tasks they are designed for, while in the field of bioinformatics, each tool is ideally suited for efficient development of software systems capable of dealing with high throughput data.

1.4.4 Bioinformatics software development

The field of bioinformatics can be considered a unique area of software development due to a number of factors. First, the majority of bioinformaticians do not have a computer sciences degree, but instead have a training background from within the biological sciences. This phenomenon has had generally a negative effect because without formalised training in programming and software development the majority of those involved in bioinformatics have acquired their programming skills from

short courses or in a self taught manner. While this mode of learning is a perfectly adequate way to gain the ability to start developing software, it rarely equips the new developer with an appreciation of how to create very high quality software solutions. It would be foolish however to believe all the challenges in bioinformatics could simply be overcome if all the work was done by computer scientists and professional software developers.

The second unique feature of software development in the field of bioinformatics is the speed at which the field is evolving, new techniques and analysis methods are constantly being developed and the amounts of data being generated by experiments continue to increase. This means bioinformatics has high demands of software, it has to be developed quickly, be tolerant of changes in things like file formats and be able to cope with very large volumes of data. The setting in which the majority of bioinformatics is conducted provides even more challenges, since most bioinformatics research is conducted in an academic setting where a ‘pressure to publish’ mentality exists. Academic research is highly competitive, which means that most software is developed only as far as is necessary to publish results and not instead developed a little further in order to become fully finished software that would then become useful to other people. Due to limited resources in academia, much of bioinformatics software is developed by small teams (often a single developer) rather than by large teams as is common in a commercial setting. This situation means any software development processes applied to the field of bioinformatics would need to be equally applicable for very small development teams as well as larger teams.

There is a great need in bioinformatics software development for a process to follow that will ensure the production of high quality code that will produce reliable results in a timely fashion. However the application of ‘heavyweight’ development processes such as the waterfall process to bioinformatics projects is inappropriate,

not just because the analysis and documentation phases require a great deal of effort, which would take a large amount of time due to small team sizes but also because of the fast paced nature at which the field is changing and as discussed above, the waterfall process does not allow for change once a design is agreed upon.

The ability of agile processes to respond to change makes them far more suitable for bioinformatics software development. In addition, agile processes should scale well to the smaller development teams typical in bioinformatics. Agile approaches are also geared to speed, flexibility and quality, which are all very important for bioinformatics development.

1.4.5 Bioinformatics software development review

The subject of software quality and development processes in the field of bioinformatics is gradually attracting more and more attention as the importance of bioinformatics increases due to the heavy reliance modern biological research places on it. There have been a handful of publications concerning the subject of software development and best practices in the field of bioinformatics, although more needs to be done to raise awareness of what is a very important issue for modern scientific research.

Baxter *et al* in 2006 (Baxter *et al.*, 2006) were the first to publish a set of best practices for scientific software development. The authors begin by defining a successful software project as a code base that produces consistent, reproducible results, is usable and useful, can be easily maintained and updated and has a reasonable shelf life. The authors suggest five practices for successful scientific software development. The first is to implement a design process that addresses two questions; “what will the program do?” and “how will the results produced by the program be verified?”. Through answering the question of “what will the program

do?” developers should be able to select the appropriate technology or programming language to tackle the problem, rather than making this decision simply based on the experience of the development team. This is an important step in a scientific software project and the authors point out, with good examples, how different programming languages have clear advantages for specific tasks. More importantly, they describe how selection of the wrong technology or language will most likely result in additional work for the development team. Assessment of the available technologies or programming languages at this early stage in the development process can also save time by identifying preexisting solutions or components that can be ‘plugged in’.

The second question of “how will the results produced by the program be verified?” is very important for the creation of software which produces consistent and reproducible results as the development of a test plan will ensure the production of high quality software. The authors also highlight the importance of planning the usability of the software as they state this should be a higher priority in scientific software development. The second practice presented in the paper is the use of quality control processes, such as testing, version control and recognising and tracking program bugs. They point out that testing identifies bugs and enables these to be solved early before problems arise. Version control is put forward as critical for the tracking and tying together of software and results. Finally, the authors believe that the process of tracking program bugs should be encouraged, and managed by appropriate tools. The third suggested best practice is the use of data standards, such as accepted data formats, ontologies and the use of standardised data repositories such as ArrayExpress (Brazma *et al.*, 2003). Through adhering to these data standards whenever possible, the authors suggest this will enable accurate and efficient code development and reduce user and peer reviewer frustration. The final

best practice put forward in this paper is the incorporation of project management. In the author's opinion, the best project management method is the agile approach of Scrum, and they also suggest the implementation of project web sites or wikis as being highly beneficial for communication and management. The authors' of the paper conclude by citing two examples of successful scientific software development projects; the cancer Biomedical Informatics Grid (caBIG) (von Eschenbach and Buetow, 2007) and the Bioconductor project (Gentleman *et al.*, 2004).

Another paper on the topic of bioinformatics best practices was published in 2009 by W. S. Noble (Noble, 2009). This paper attempted to describe a clear strategy for performing high quality bioinformatics experiments. Although the issues raised in the paper, such as file and directory organisation, may seem extremely tedious to many bioinformaticians, having a well formed plan for performing bioinformatics experiments is essential for generating reliable and reproducible results. The core motivating principle for performing research in this way is that "someone" should be able to look at the files that have been generated during an experiment and simply and easily work out what was done and why. The author describes that the "someone" could be any number of different people such as collaborators, researchers wanting to repeat your work, or more commonly than not, the person who actually performed the experiments, having to revisit the work after a period of time. Nobel puts forward an additional principle which he believes applies to computational biology experiments, that some part (or all) of an experiment is highly likely to need repeating at some point. The need to repeat work may be due to the discovery of mistakes in the original analysis, or new data becoming available, but whatever the reason, it is far easier to go about repeating an experiment having used a good strategy for carrying out the work in the first place.

The paper goes on to describe an ideal strategy for performing a bioinformatics

experiment, beginning with the organisation of the directories and files. The principle put forward is a simple one; start with a single top level directory, within that directory should be placed directories for data, results, source code that needs to be compiled (src), scripts and compiled (bin) and documents. Data and results should be placed in dated directories corresponding to when the files were created. The contents of the src directory should be source code for compiled programs and the bin directory should contain simple scripts, finally the documents folder should contain a draft manuscript for the work being carried out. The second practice recommended by Nobel is the use of a bioinformatics laboratory notebook. Dated entries in the lab book should give a detailed account of each step carried out, as well as results and interpretations of the work. Online solutions such as blogs or wikis provide the ideal solution for a bioinformatics lab book as they can easily be updated and viewed by external collaborators. The practice of handling and preventing errors is also discussed in the paper, with the author recommending protocols to ensure errors are dealt with correctly: numerous checks should be included in even simple programs; scripts should be used to ensure the integrity of data; and on the discovery of an error, the correct action should always be to terminate the program. The final practice discussed for use in the workflow of a good bioinformatics experiment is version control, because as the author states, version control serves a form of backup for work and results. Version control also provides a historical record of the work carried out, making it easier to understand the changes that have gone on during an experiment. Finally, version control simplifies collaboration, allowing numerous people to work on the same set of files without fear of overwriting each other's work. This paper puts forward the idea that the logistics of efficiently performing accurate, reproducible computational experiments is a subject worthy of consideration and discussion.

Dudley *et al* (Dudley and Butte, 2009) in their paper of 2009 present another set of guidelines for effective bioinformatics programming. In contrast to the papers described above, the authors here concentrated on the specific tools for a variety of bioinformatics tasks. The first bioinformatics task they discuss in the paper is the day to day programming which is necessary in all forms of bioinformatics. The authors point out that modern interpreted scripted languages such as Perl, Python and Ruby are among the best choices for general bioinformatics programming work due to the speed at which solutions can be created and the availability of libraries of code available for each. Beyond programming languages, the authors recommend that bioinformaticians should also become familiar with a web server system such as Apache, based on the clear importance of web tools to the bioinformatics community. The authors go on to discuss the importance of using freely available source code that solves common bioinformatics problems. The authors recommend a number of frameworks of source code, such as bioPerl, bioJava, bioPython, bioRuby and bioconductor, as their use prevents developers from wasting valuable time ‘reinventing the wheel’. In common with the previous two papers, Dudley *et al* also discuss the importance of good documentation and version control of project related files. The form of documentation focused on in this paper is the annotation of source code because, as they state, good documentation is key to making good sense of code. Again with the focus on recommending the correct tools for a task, the authors suggest the use of automated documentation software including Doxygen (www.doxygen.org), JavaDoc (www.oracle.com/technetwork/java/) and PyDoc (www.python.org/). On the subject of version control systems (VCS), the authors concentrate again on the specific tools available to bioinformatics software developers mentioning CVS, Subversion and Git as VCS, and a number of related tools such as TortiseSVN and SCPlugin which integrate VCS functions into operating

systems. The use of free online services like SourceForge (sourceforge.net) or GitHub (github.com) for hosting a VCS is also a very sensible addition to their discussion. The authors then go on to raise some very good points about data management, an important issue not thoroughly touched upon in the previous papers. They put forward the idea that the use of flat files for bioinformatics is outdated and that relational databases are far better suited to the problems that bioinformatics poses. The integration of data from a variety of sources is far easier using a relational database than using flat files, and computationally they offer far more elegant and efficient solutions for interacting with data, such as object relational mapping frameworks.

Object relation mapping (ORM) is a technique for intelligently mapping the tables and rows of a relational database to programming objects, in order to provide consistent access to a data model. ORM frameworks are available for many popular programming language and database combinations, such as: Perl's DBIx::Class (Trout, 2012), Java's HIBERNATE (hibernate.org, 2012), Python's Storm (Canonical, 2012) or Ruby's Active Record (rubyonrails.org, 2008). These example ORMs all provide developers with routines for common database interactions, including: creating, reading, updating and deleting database entries. They differ slightly however in how they generate an object layer to a database, with for example Perl's DBIx::Class ORM using the structured query language definition of an existing database to create objects, whereas, Python's Storm ORM creates a database schema from user defined object classes. In both of these cases the developer only needs to define a data model in a single location which is one of the main advantages ORM frameworks provide. Having a single consistent location for the underlying data model in an application avoids having to make changes in at least two locations at the database level and the programatic level wherever the data

model is accessed thus making it much harder to introduce mistakes.

An interesting paper by Kane *et al* (Kane *et al.*, 2006) presented a different aspect of the use of best practices in bioinformatics software development, specifically the use of agile development practices. The authors identified a number of academic and commercial organisations already using agile development approaches for scientific software development. They surveyed those involved in each of the agile projects to gain information on two main levels. First, basic information including the agile practices that were being used in the project; the development process used previous to agile; and the number of people working on the project. The more interesting second stage of the study collected information on the feelings and experiences of the agile users.

The authors were able to collect information on six organisations (five academic and one commercial) using agile practices for scientific software development. The first finding they made was that the size of the teams undertaking the work were all relatively small, which will be unsurprising for anyone with experience in the field of bioinformatics. On the use of software development practices before implementing agile practices, two organisations had used no formal process, four had been using more heavy weight process and one had used agile for the beginning of the project. The survey of the agile practices used in the different projects revealed a very useful set of common agile practices that were used across all projects:

- Automated unit tests
- Continuous integration
- Feature backlog
- Refactoring

- Open workspace

From the more in depth interviews concerning the feelings and experiences of using an agile approach in a scientific software development setting, the developers reported better interaction with the scientists involved with the project. The authors suggest that a common attitude amongst scientists is that software development is an ancillary task that should not have too much time devoted to it. However, they found that using agile created a greater level of shared responsibility, improving the way the software development process is perceived. The authors also report that the projects found they had improved quality, flexibility and maintainability for using agile processes, concluding that agile methods are well suited to scientific software development.

All of these papers highlight a great need for a more uniform use of software development best practices in the fields of scientific software development such as bioinformatics, and they also all suggest very valid approaches for achieving this aim.

1.5 Aims

The first aim of this thesis was to develop a framework of bioinformatics software to meet the informatics requirements of our laboratory's aCGH profiling projects, including the management and analysis of the very large volumes of data generated. An important requirement was for the system to be highly scalable, in order to respond to the demands of ever increasing project sizes. Beyond the development of the framework, the thesis would aim to test the system by applying it to a large aCGH ovarian tumour profiling project, testing the hypothesis that the profile of somatic genetic alterations is a determinant of patient survival.

Throughout this thesis the primary hypothesis being tested was that by approaching bioinformatics software development in a more formalised manor by using agile software development practices as well as other development best practices this would result in software which is of a much higher standard (reproducible, reliable and maintainable) compared to software developed using a traditional bioinformatics *ad hoc* approach.

In chapter 2 of this thesis, I will describe a Laboratory Information Management System (LIMS) that manages and records all aspects of microarray manufacture and experimentation. In chapter 3, I will describe the implementation of object orientation programming techniques on new and existing source code in the laboratory, including a description of the suite of Perl analysis objects developed to provide a flexible way quickly to construct data pipelines for quality control normalisation and analysis using packages from the Bioconductor project. Chapter 4 describes the application of these software to the analysis of aCGH data from 94 ovarian tumour samples, in order to detect copy number changes that are associated with patient survival. The final chapter of the thesis, chapter 5, contains a discussion of the various issues raised by this work.

CHAPTER 2

Extending a microarray manufacture laboratory information management system to support aCGH experiments

2.1 Background

As discussed in section 1.3.5, a high quality Laboratory Information Management System (LIMS) is an essential component in the informatics infrastructure of any laboratory wishing to carry out research using high throughput techniques. The Translational Research Laboratory (TRL) had already taken the decision to create a bespoke LIMS solution before the beginning of this thesis, to support their microarray manufacturing work. At the time this project started, there was user demand for greater functionality in the LIMS, in order to support the aCGH experiment workflow and subsequent data analysis. This chapter describes the

extensive work carried out to create a LIMS capable of supporting the demands of high throughput aCGH projects.

2.1.1 The existing in house LIMS

The LIMS solution that was already under development in the laboratory was named ArrayPipeLine; at the beginning of this thesis the ArrayPipeline LIMS was supporting the manufacture of the in house BAC aCGH microarray platform. The ArrayPipeLine LIMS contained features to assist in all aspects of microarray manufacture, such as methods for dealing with the microtitre plates used routinely for samples and array features, management of interactions with laboratory equipment such as robotic arrayers and tracking of microarray designs and details on the spotted features.

The microtiter plate functionality of the LIMS was a key feature because of their importance in many aspects of high-throughput experiments, being used for storage and manipulation of all biological material used. The LIMS included methods for creating and manipulating microtitre plates of different formats including 96, 384 and 1536 well plates, as well as individual tubes. The standard features available for all plates included methods for adding samples to multiple and individual wells, and the ability to identify plate contents. Some of the most useful features concerned whole plate manipulations like the ability to perform plate to plate transfers, join plates of a similar format into a larger format, or combine plates. The LIMS also had support for storing information on the processing steps required for the manufacture of microarray slides. Most importantly for the manufacture of microarrays is the ability accurately to identify each spotted feature on the microarray slide; for this the ArrayPipeLine LIMS was able to record the design of every microarray manufactured by the laboratory. The ArrayPipeline LIMS also included specific functionality to

support the manufacture of aCGH microarrays such as the ability to store and retrieve the precise genomic location of the BAC clones used as features to create the array.

The laboratory based users requested new features in the ArrayPipeLine LIMS to support the laboratory's aCGH experiment workflow and support the demands of high throughput tumour profiling projects. In order to respond to these requests required a large amount of work in three areas of the existing LIMS. First, the structure of the source code in the existing LIMS required significant improvements in order to make the code base more modular, to reduce redundancy and to increase code re-use for greater efficiency. Second, the backend storage solution (a MySQL relational database) would need to be expanded to incorporate the extra data generated by the aCGH experiment workflow and subsequent analysis. Finally, expansion of the user interface would be required, in order to provide functionality for users to interact with their experimental data. The rest of this chapter will describe the challenges involved in extending the ArrayPipeline LIMS to be able to cope with the broadening requirements posed and the solutions that were developed to satisfy those requirements.

2.1.2 Initial ArrayPipeLine LIMS code base

The initial implementation of the ArrayPipeLine LIMS was not optimal for easy expansion in functionality, in addition to many other issues concerning its development. The LIMS was developed using a simple system of a web-based user interfaces, driven by very long Perl Common Gateway Interface (CGI) scripts, that directly interacted with the underlying MySQL relational database. This system created high levels of redundancy between scripts in performing common tasks, and redundancy in the code created very long scripts which were difficult to maintain and

easy to introduce bugs into. The direct access of individual scripts to the underlying database gave no insulation to changes in the underlying data structure, which meant that any changes to the ArrayPipeLine LIMS database could potentially break any scripts which accessed those data. Also the code was not contained in a version control system, which made it difficult for the two developers in the bioinformatics team to work on the same pieces of code without destroying each others changes. In addition, a lack of version tracking meant that it was impossible to trace the origin of bugs in the code.

2.1.3 Extending the LIMS database for biological samples

The initial priority for expansion of the existing manufacturing LIMS was for the tracking and management of the information from the various biological samples that a high throughput laboratory performing aCGH analysis might encounter. It was also essential that the detailed information recorded was accurate in order to avoid mistakes in interpreting the resulting data.

Tumour samples

Owing to the fact that the primary focus of the laboratory was the further understanding of ovarian cancer, one of the most important biological sample types are patient derived tumour samples. There is a lot of information associated with tumour samples and capturing as much information as possible is important for downstream analysis. This information includes clinical features of the tumour such as stage and grade. The stage of a tumour is a measure of how advanced the disease is and is based on a number of factors, including; the size of the tumour; whether lymph nodes contain cancer; and whether the cancer has spread from the original site to other parts of the body. The system for staging in ovarian cancer

comes from the International Federation of Obstetricians and Gynaecologists (FIGO) (Benedet *et al.*, 2000). Staging ranges from one to four, in which stage one defines a tumour confined to the ovary or ovaries, while stage four indicates evidence of distant metastasis beyond the peritoneal cavity (Benedet *et al.*, 2000). The grade of a tumour is a measure of how abnormal the cancer cells look microscopically. Although the system of tumour grading can differ between types of cancer, typically the grade of a tumour is scored on a scale between one and four with one being a well-differentiated tumour and four being a undifferentiated tumour (Benedet *et al.*, 2000). Tumour stage and grade details are very important to record as this information can be used to great effect in extracting more information from downstream analysis, therefore having a system where this information can be recorded accurately and retrieved easily is very important.

When recording the tumour sample information it is also important to consider that multiple tumour samples may originate from the same patient, for instance; multiple samples can be taken from a single tumour; from a recurrence of the tumour; from sites of metastasis; and finally ovarian cancer patients can have bilateral disease, meaning that there are tumours on both ovaries. For this reason it is necessary to include the patient when modelling the data, allowing the system to record basic information about age and survival time only once at the level of the patient rather than multiple times at the level of the tumour and thus avoid de-normalising the database by introducing redundancy. Additionally the sample collections used in a aCGH profiling project can include other data, such as; the treatment provided; the subsequent response to the treatment; and detailed epidemiological data.

Normal samples

Along with the test sample, the aCGH experiment requires a normal sample that serves as a reference. Obviously it is equally important to record detailed information on the samples used for reference in all experiments. Reference samples typically come from one of two sources; either a pool of normal DNA, or a matched normal DNA sample from the patient.

External samples

The ArrayPipeLine LIMS also required the ability to deal with samples with minimal additional information and import experimental data from such samples that may come from external collaborators where additional information was limited. The solution was to add to the data model what is referred to in the ArrayPipeLine LIMS as an ‘external sample’. External samples can be incorporated into the ArrayPipeLine LIMS with minimal information about the sample itself and the experimental procedures used to generate the sample. This allows users of ArrayPipeLine to apply the features of the LIMS, such as visualisation tools or analysis pipelines, to externally sourced data.

Sample collections

To enable the laboratory to deal efficiently with groups of samples and allow researchers quickly to retrieve results from the experiments they are interested in, it was necessary also to consider sample collections in the data model. A sample collection is a very simple concept, comprising a group of biological samples associated in some respect for the convenience of experimentation or analysis. A sample collection in the ArrayPipeLine can simply represent an existing internal or external sample collection, or they can be used in more useful applications such as for

creating cleaned sample collections where a quality control process has been applied to remove poor quality samples. Through the inclusion of tools for the recording and retrieval of sample collections in the ArrayPipeLine LIMS, the system and therefore the laboratory, is also capable of performing experiments on many different forms of cancer or any other samples where aCGH analysis may provide insight.

Importantly the sample information system in a high throughput aCGH profiling LIMS needs to be flexible enough to record any potentially valuable additional data.

2.1.4 Extending the LIMS database for aCGH experiments

At the start of this thesis the ArrayPipeLine LIMS was well featured to support the recording of all the experimental details required for the manufacture of custom BAC microarrays. The features included recording procedure details from processes such as DNA amplification, DNA purification and microarray printing as well as recording basic information on the microarray features and the microarray design. However, in order for the LIMS fully to support the management and analysis of high throughput aCGH projects, additional support was required that would allow details to be recorded for the entire aCGH experimental process.

Fluorescent labelling of DNA samples

Central to the technique of competitive hybridisation is the differential labelling of the samples to be co-hybridised, most commonly achieved by attaching different fluorescent dyes, such as cyanine or Alexa Fluor, to each of the samples. The process of attaching these fluorescent dyes to the biological samples is termed labelling and was the first microarray experimental process which was added to the ArrayPipeLine LIMS in order to support recording of aCGH experiments. Essential information to be recorded by the LIMS includes the identity of the sample being

labelled, information about the type of dye used and details of the labelling protocol. Additionally, measures of the efficiency of the labelling procedure can be generated and recorded which can subsequently be used for quality control purposes.

Hybridisation reaction

Once all the samples, which are to be combined for the experiment, have been successfully labelled, information then needs to be recorded on the hybridisation procedure in which the differentially labelled samples are co-hybridised onto the surface of the microarray slide. At this stage of the experiment, it is necessary to record the identity of the two or three labelled samples which are to be hybridised, together with details of the microarray slide and the hybridisation protocol. Details of the design of the microarray were already recorded in the manufacturing LIMS, along with information regarding the batch in which the microarray slide was created. This is extremely useful for ensuring quality control of the microarray slide batches used for a project.

Image acquisition

Following hybridisation, the microarray slide is scanned by multiple lasers, and light emitted from the excited fluorescent dyes is collected by a photo-multiplier tube. This process is called scanning, and generates a series of TIFF images of the microarray slide, one for each fluorescent dye used in the experiment. The complex scanning process involves numerous settings that need to be recorded for quality control purposes, to ensure consistent data capture across projects and to enable the successful replication of experiments.

Image analysis

The image analysis step extracts fluorescent signal intensities at each feature of the microarray and involves numerous parameters, all of which need to be recorded for quality control purposes. The output of image analysis software is a single results file containing the signal intensity values for each feature and associated data such as quality control metrics. The large amount of data contained within these files, typically 30,000 rows x 20 fields, means they are very large (80Mb) and as a result file Input/Output (IO) operations on a large number of files can be time consuming.

Recording the results of the image analysis within the LIMS would allow more rapid retrieval of results, and would enable detailed interrogation of individual experimental data than would be possible with a flat-file based system. An additional consideration comes about because each image analysis software solution provides its own data output format, which means that bespoke database tables are required for each software package in order to capture any unique information it provides.

Microarray results files

A single aCGH experiment gives rise to a minimum of two TIFF image files each time the microarray slide is scanned and a data file of intensity values each time an image analysis is performed. This means that an aCGH project aiming to profile a thousand samples will, depending on the experiment design used, generate a minimum of three thousand experimental results files. Once the important data have been extracted from these files and recorded in the LIMS, there is a strong case for not storing these files. However this approach has a number of drawbacks which include the inability to reanalyse older scan images using new or updated image analysis software to

extract better quality results. If an error was detected in the experiment pipeline it would not be possible to go back to track the source of a potential problem, thus resulting in truncation of the audit trail. Finally archiving all the generated experimental files creates an additional layer of backup which is a positive impact on the system, as multiple backup sources provide greater security against data loss. The financial cost of computer storage is relatively inexpensive when compared to the cost of samples, reagents and time required to re-run an aCGH experiment. For all of these reasons the decision was made to include in the ArrayPipeLine LIMS a system that would allow the archival of all the experimental results files created.

2.1.5 Extending the LIMS database for recording analysis steps

The details of each of the numerous pre-processing steps that are required to be carried out on aCGH data need to be accurately recorded, both to ensure consistency between the experiments and the reproducibility of results.

Analysis steps and parameters

Microarray data analysis packages, such as those available from the Bioconductor project, typically have a number of parameters that allow users to tune the performance of their analysis. It is therefore very important that when using a published analysis package or a bespoke solution that all the possible parameter settings are recorded. In order to reduce redundancy in repeatedly recording common settings, parameters should be recorded as parameter sets, thus allowing commonly used parameter sets to be quickly and easily accessed and used again.

Analysis pipeline

The term ‘analysis pipeline’ refers to a series of analysis steps performed in sequence, with the output of one analysis serving as the input for the next. An analysis pipeline is a very useful way of generating higher level results from raw data using a series of different analyses. Because the user does not have to run each analysis individually or handle the intermediate data, it makes analysis pipelines very powerful tools. An analysis pipeline is in essence a simpler interface to a collection of more complex analysis processes that still provides the ability to easily re-configure and re-run the analysis.

It is very important to be able accurately to record the analysis pipeline applied to experimental data in the database to ensure the reproducibility of results. Through recording commonly applied analysis pipelines users are able quickly to apply a standard analysis to some new data for comparison of results with previous experiments. Recording an analysis pipeline is a relatively simple task, but it is vital to be able to track exactly which analyses have been applied to experimental data and in which order, as this can greatly affect the resulting data.

Processed data

In theory, the processed data generated as a result of applying an analysis pipeline to some data does not need to be stored in the LIMS database, as this would increase the amount of redundant data in the database. Instead, all the processed data values can simply be created from the raw intensity data already in the database by applying the analysis pipeline again. This approach would be possible if the required analysis process was fast enough to be performed on the fly, however the statistical analysis methods used to analyse a single aCGH experiment can take

several minutes. This is too time consuming for a web-based visualisation of the experimental data and clearly unsuitable for larger scale, project-wide analyses. It is therefore necessary to record the processed data generated by analysis pipelines in the LIMS database.

2.1.6 Extending the ArrayPipeLine LIMS GUI

In order to support the additional functionality discussed required expansion of the LIMS user interface that allows users to be able easily to take advantage of the new features. The LIMS therefore needed interfaces that enabled users to record the details of projects and samples as well as interfaces which allowed users to record the details of processes involved in carrying out an aCGH experiment. Along side interfaces to record data was an obvious requirement for easy to use interfaces which allowed the extraction of data from the LIMS for the purpose of quality control and further analysis.

2.2 Methods and implementation

The extended Array Pipeline LIMS was built upon software standards in the field of bioinformatics using a three tier architecture, consisting of; the data layer; the application layer or application programming interface (API) and finally the user interface layer (UI). The data layer typically takes the form of a database, which offers the best solution for fast and accurate storage and retrieval of information. The application logic layer acts between the data layer and user interface, providing the functionality of the system by interpreting all requests from the user interface and interacting with the data layer. The user interface provides the software user with an easy to use method for interacting with the application, by providing

simple to use functionality for requesting and receiving data. Given this modularity in the application, the methods and technologies that were used to extend the ArrayPipeLine LIMS can be split into three sections.

The use of freely available standards in all three tiers means the ArrayPipeline LIMS is platform independent and highly extendable.

2.2.1 Database

The Array Pipeline LIMS uses the relational database management system MySQL for its data layer. MySQL is a free open source Relational DataBase Management System (RDBMS). The MySQL RDBMS is very widely used in bioinformatics and was chosen for this project because of a number of features; MySQL is very easy to use but with support for very advanced features such as programming interfaces for many languages, such as C, Perl and Java; the RDBMS is highly portable being capable of running on many operating system; MySQL is fully networked, meaning databases can be accessed from anywhere on the Internet, so data can easily be shared; MySQL has easy to configure security features which enable users to protect the data in a database; and finally the MySQL community is very active and responsive in providing support (DuBois, 2008).

Extending the data layer

In order for the ArrayPipeLine LIMS to support the new required functionality that was described in sections 2.1.3, 2.1.4 and 2.1.5 required significant modification of both the data layer and the application layer. The MySQL data layer of the ArrayPipeLine application required the design and addition of a number of new database tables that would be able to capture all the necessary information and data relationships. The design process used to extend the ArrayPipeLine database

involved the following steps;

- Requirement analysis: The first step in extending the database was to perform a requirement analysis to identify exactly what the extended database should support and all the different forms of data that this would involve. The requirement gathering step involved identifying the numerous sources of possible data from high throughput aCGH projects that would require storage, such as the file formats exported from image analysis software. The requirement analysis process also involved talking to the laboratory based customers of the ArrayPipeLine LIMS to discuss what information they required recording, such as experimental procedures. A thorough approach to this task enabled us to identify all the most useful data fields to record which would be essential for tasks such as quality control and downstream analysis, and at the same time identify those data fields that were redundant or could be easily calculated from existing data.
- Entity Relationship (ER) diagrams: Entity Relationship (ER) diagrams allow database developers to visualise the relationship between tables in a database design. ER diagrams were used in the next step of the design process for extending the ArrayPipeLine database, in order to identify many to many relationships in the preliminary design. Many to many relationships are not supported by most database systems and thus have to be resolved through the addition of a table which further explains the relationship of the data involved.
- Normalisation: Database normalization is the practice of optimising a database to avoid redundancy and prevent inconsistencies, the idea of database normalisation was first proposed by E. F. Codd (Codd, 1970). The removal of redundant data from a database is important as redundant data wastes disk

space. Redundant data can also lead to the introduction of inconsistencies in the database. If the same data exists in multiple locations in a database, then every update or delete operation would need to be successfully completed in all locations in the database, a situation which is liable to the introduction of errors in the data.

The advantage of having a highly normalised data schema is that information is stored in one place, reducing the possibility of inconsistent data following update or delete operations. Furthermore, highly-normalised data schemas in general are closer conceptually to object-oriented schemas because the object-oriented goals of promoting high cohesion and loose coupling between classes results in similar solutions. This generally makes it easier to map programming objects to a normalised data schema (Ambler, 2010).

Database normalisation is performed in terms of normal forms, each form addresses a situation in the database that will cause problems. The normal forms build on each other, so a database can not be normalised to third normal form without also being normalised to first and second normal form. Further explanation of the three most common normal forms follows:

- First normal form (Codd, 1970) states that a table should not use fields which contain multiple similar values, instead a new table with the multivalued information and the primary key of the original table should be created.
- Second normal form (Codd, 1971b) states that a table should be in first normal form and records should not depend on anything other than a table's primary key (a compound key, if necessary). Where a table is not in second normal form the fields that are not dependant on the primary

key should be moved into a new table with a primary key they depend on.

- Third normal form (Codd, 1971a) states that a table should be in second normal form and values in a record that are not part of that record's key do not belong in the table. In general, any time the contents of a group of fields may apply to more than a single record in the table, consider placing those fields in a separate table. A table is in third normal form if it is in second normal form and no non-key fields depend on a field(s) that is not the primary key.

Fourth (Fagin, 1977) and fifth (Fagin, 1979) normal forms take the normalisation process further by going beyond the functional dependencies that the first three normal forms deal with. However normalising a database up to third normal form is the highest level necessary for most applications, in some cases normalisation steps beyond third normal form can even penalise retrieval, since data which may have been retrievable from one record in an unnormalised design may have to be retrieved from several records in the fully normalised form (Kent, 1983).

Database normalisation was applied to the design process, with the initial designs for extending the ArrayPipeLine database normalised were possible to third normal form.

- Optimisation of the design: The design up to this point is database independent and could be implemented using any relational database software, however in order to make the resulting database fast and efficient the design required additional database software specific information. The additional information included details of the data types used for each field of data;

using the most appropriate data type can save a large amount of memory over millions of entries, and hence impact upon the memory requirements of the database. The addition of database indexes was also very important in creating responsive database tables, greatly improving the speed of data retrieval at the cost of slower writes and increased storage space.

2.2.2 Creating an object oriented Perl API

The redesign of the original ArrayPipeLine LIMS source code to apply object oriented techniques was undertaken using the programming language Perl which was chosen based on a number of features which make it well suited for developing bioinformatics applications, such as: built in support for text processing; advanced, yet simple to use database support; and a very active and helpful developer community which includes the Comprehensive Perl Archive Network (CPAN) a large repository of third party modules containing reusable code solutions for many common development tasks. The original code base had been developed in Perl, therefore using Perl also made recoding sections of existing code much easier without the need to change any programming syntax.

To assist in the object oriented source code redesign and future development, a source code versioning repository for the bioinformatics team of developers was setup using the highly popular open source Concurrent Versioning System (CVS) project. A repository was created on the laboratory's local server and all subsequently developed source code was added to, and maintained by, this repository.

2.2.3 User interface layer

The UI layer of the The Array Pipeline LIMS was implemented using a system of dynamic content web pages that are created using common gateway interface (CGI) programming scripts, written in Perl. The user interface was tailored towards the requirements of laboratory based users, ensuring it was easy to use for the entry and viewing of experimental data. The web based user interface also reinforces the portability of the ArrayPipeline LIMS as it will work with any browser on any operating system without the requirement of installing additional software.

2.2.4 Agile software development

The extension of the ArrayPipeLine LIMS was undertaken using a number of agile development practices, including user stories, an onsite customer and constant integration.

2.3 Results

2.3.1 Requirements gathering

The requirements gathering process involved meeting with and speaking to three groups of people within the laboratory, the current users of the ArrayPipeline LIMS, the project managers that were currently or planning to use the TRL aCGH facility and the other software developers working on the ArrayPipeLine LIMS. Through meeting with these three different groups of people a number of times, the requirements that are detailed in the background section of this chapter were identified. The requirement gathering process used an unformalised system, similar to the agile approach of customer stories. Users would simply describe what they

needed from the system using language and terminology they were comfortable with. By using an unformalised system it was easier to respond to changes in their requirements.

In summary the main requirements gathered were:

- Improvement of the existing LIMS code base. The improvement of the existing ArrayPipeLine LIMS source code base was a requirement identified by the developers of the LIMS. The LIMS source code was required to be more modular, better documented and more robust in order to make expansion of the LIMS easier.
- Provide support for recording the details of new and existing sample collections. The requirement for recording the details of new and existing sample collections came from the current LIMS users and the project managers. As all aCGH experiments involve samples, it was identified as very important for the LIMS to be able to accurately and reliably record and recall the identity of samples used in the laboratory.
- Record details of the individual steps of the aCGH experimental process. The LIMS users required the ability to record and store the details of each of the aCGH experimental processes. The LIMS users requested a system that would allow them to easily record and recall the details of the multiple experimental steps required as part of an aCGH experiment. The LIMS developers also required the details of the experiment steps to be recorded to enable them to provide quality control information to the LIMS users. This information was also essential in downstream analysis.
- Provide a system capable of dealing with high throughput aCGH projects. The project managers identified the requirement for the ArrayPipeLine LIMS

to be able to cope with high throughput aCGH projects involving thousands of samples. This requirement was based on information on the number of samples that were likely to pass through the laboratory for aCGH profiling in the coming months and years.

- Extend the ArrayPipeLine LIMS GUI to support all the new functionality The LIMS users identified the need for the LIMS to provide easy to use interface for recording samples and aCGH experiments. The LIMS users were not comfortable simply using a database interface to record the new information the LIMS would now allow, they instead required a more user friendly way to interact with the new functionality.
- Record details of the analysis steps performed on the aCGH experimental results. The developers of the LIMS required the ability to record all the detailed information associated with processing aCGH experimental data. As each analysis step of which there are multiple can involve numerous parameters, it was important for the developers that all of this information was recorded for data reproducibility.

2.3.2 Object orientation of the ArrayPipeLine LIMS source code

Work completed in combination with my supervisor on the problems with the original LIMS code base lead to the creation of an API for development of the ArrayPipeline LIMS. Using object oriented programming techniques we were able to create a generic toolkit of Perl modules that would enable developers to create useful LIMS solutions in a way which was simple to use and required minimal effort. The toolkit includes a number of methods that extend the functionality of two Perl modules; DBI.pm (Bunce, 2012) the standard database interface module for Perl

and CGI.pm (Stein, 2007), the common gateway interface class for Perl. Alongside Perl, the toolkit has been developed using two other standards in bioinformatics application development: the open source database MySQL and the freely available Apache web server. Although we have chosen to use MySQL and Apache for development of the toolkit, it should be possible to use any database based on the Structured Query Language (SQL) and any CGI capable web server.

ArrayPipeLine LIMS framework

The ArrayPipeLine LIMS software framework is build upon four generic object-oriented Perl modules that all share the ‘LIMS’ namespace on CPAN, LIMS::Base, LIMS::Database::Util, LIMS::Web::Interface and LIMS::Controller. Implementation specific functionality has been provided using the object oriented technique of inheritance to create sub-classes of the core LIMS::Controller class for example LIMS::ArrayPipeLine.

LIMS::Base is the base Perl class for the LIMS suite of object oriented Perl modules, implementing a number of higher level generic methods important for generating the user interface and interacting with the underlying database. LIMS::Base methods provide functionality in two main areas, configuration settings and error handling. The ‘load_config’ method is one example of the available configuration methods, it parses configuration files into a Perl data structure allowing for simple query and retrieval. The ‘get_error_string’ method of LIMS::Base is an example of the available error handling methods, the method when called, returns all the accumulated errors from the multiple possible sources in a session, in a formatted string ready for output.

LIMS::Database::Util is the object-oriented Perl module that serves as an object layer or DataBase Interface (DBI) for any LIMS database. The module inherits from

LIMS::Base and provides automation for DBI services required by a LIMS database, enabling rapid development of Perl CGI scripts. The DBI methods in the module are wrappers for DBI.pm calls, catching possible errors so that the way they are reported can be controlled in the CGI script. The module contains nine methods for returning databases results in different data structures in order to suit the differing requirements of the developer, the methods in LIMS::Database::Util are able to return the following Perl data structures: scalar, array, two dimensional array, hash, two dimensional hashes and binary objects. The module also contains two methods for inserting data into a database, and a single method for updating values already present in a database. The LIMS::Database::Util methods 'insert_into_table' and 'insert_with_placeholders' both insert new data into a database, the first method 'insert_into_table' is a simple method that only requires the name of the table to insert the data into and the values to insert in a pre-formatted string. The second method 'insert_with_placeholders' is more sophisticated, by using a two dimensional array for input it allows users to easily insert multiple records in a table using a single call to the method. Unlike other methods, the method will also 'kill the pipeline' if any errors are caught, or commit the new records to the database upon successful completion. The method can return either a list of the newly inserted ids created by insert statement or it can return the number of inserted rows. Where update operations are required 'simple_update_placeholders' method provides an easy to use method, single values in a selected table are updated when provided with an updated value and 'WHERE' clause.

The development of multi-layered applications such as LIMS is made more difficult because errors can be generated at many different levels of the application, from the development language Perl, the database, the database interface, the web sever or the common gateway interface. Correctly tracking and handling these

errors is essential in creating a robust application. Three methods, 'db_error', 'standard_error' and 'any_error' handle the errors in the LIMS::Database::Util module, and the 'kill_pipeline' method prints them out upon killing the script; 'db_error' returns any database (DBI) errors that have been caught; 'standard_error' can be used to set any error/complaint in a CGI script, or returns any standard error that has already been set; while 'any_error' returns true if errors of any type have been caught. The 'is_unrepentant' method can be called at any point in a script to cause the script to die if any errors are thrown, printing out all errors and issuing a rollback call to the database. When combined with any critical actions that need take place in the LIMS this method can prevent a database from becoming corrupt.

LIMS::Web::Interface is the object-oriented Perl module designed to act as a layer between a LIMS database and its web interface. It inherits from the LIMS::Base module and provides automation for services required in developing a LIMS web interface. The LIMS object provides methods for formatting the HTML layout of a new page quickly and easily; the header and sidebar HTML are both printed using a single method call to 'print_header', the page content is then added, and finally the page footer HTML is printed with a method call to 'finish' which also closes open connections to the backend database and forwards any parameters that have been set. The method, 'is_back_sensitive', which when called in an interface script, prevents the user from using the back button on their browser by rejecting an old session id. This functionality prevents multiple submission of a web page which could result in duplicate data entering the database.

The creation of correctly formatted URLs incorporating CGI parameter values is important, so the LIMS::Web::Interface module contains forwarding methods that will transfer data across multiple CGI pages. The method 'param_forward' will forward all the currently set parameters in a web page as hidden values

and ‘format_url_base_query’ formats the current user name and session identifier parameter values to append to a CGI script’s URL, ‘format_redirect_full’ generates a URL for a given script which including all required parameters. The use of formatting methods means that URLs are never hard coded in any scripts, so any change to a URL can be made in a single configuration file rather than in multiple locations across all CGI scripts.

Although the tasks fulfilled by this module are relatively simple, they are very useful as they are so frequently used in the generation of web based user interfaces. By using method calls to a module the developer reduces the number of lines of code that needs to be written which both decreases the amount of time it takes to create a new interface but also reduces the possibility of introducing errors or bugs into the code. Also using the LIMS modules allows the developer to concentrate on functionality of content rather than layout, using the header, sidebar and footer methods allow developers easily to maintain a consistent style throughout all interfaces.

LIMS::Controller is a versatile object-oriented Perl module designed to control a LIMS database and its web interface. Inheriting from the LIMS::Web::Interface and LIMS::Database::Util classes, the module provides automation for many core and advanced functions required of a web/database object layer, enabling rapid development of Perl CGI scripts. The LIMS::Controller module extends Apache and MySQL authorisation and security by providing methods for verifying the username and password of anyone attempting to access a restricted web page, using a ‘user_information’ table in the LIMS database. If this step is successful then a session is automatically generated, however if inactive for longer than the set session time the session expires and authentication is again required. The method ‘new_guest’ is similar but does not require the user parameters and does not verify login which

allows developers to quickly and easily generate non sensitive web pages suitable for a wider audience using the LIMS template. The method ‘new_script’ returns a new object without CGI/DBI or user login which is useful for rapid prototyping of new methods against the LIMS database.

The LIMS::ArrayPipeLine module inherits from LIMS::Controller, the module adds implementation specific functionality for the Translational Research Laboratory CGH-Microarray LIMS. Methods are provided for formatting the HTML layout of each page, divided into a header, sidebar and footer. A single method ‘print_trl_header’ calls methods that print the CGI header, the page title, and the sidebar, ready for the CGI script to add content to the body of the web page. At the end of a script, a modified version of the method finish will call a method to print the TRL page footer. The static HTML elements inserted by the LIMS::ArrayPipeLine module are specified in a configuration file, this removes the need to replicate sections of HTML code across all interface scripts. Removing the need to replicate sections of HTML reduces the possibility of introducing HTML errors or bugs.

Testing was undertaken to establish if it would be possible to discard the original data files by placing them into the database as binary long objects (BLOBs). However following testing we were unsatisfied that using BLOBs was a good substitute for storing the data files, we had difficulties on a number of occasions using BLOBs recreating the stored data files. Because of this we made the decision within the ArrayPipeLine LIMS to store the original experimental data files in a simple file system.

The LIMS::ArrayPipeLine module provides methods for dealing with data files, by having the directory structure set in a configuration file the object methods provide consistent paths for storing and retrieving data. This approach makes the ArrayPipeLine LIMS less susceptible to errors with file locations when compared to

a situation where the location of files is hard coded into each script which introduces more opportunities for mistakes to be introduced into the code base. The methods 'storage_path' and 'web_path' append a file name or path with either the storage path for LIMS files, or the web path to retrieve them.

The LIMS software framework enables developers to interact with databases and create interfaces in a quick and simple manner, as an example, the source code listed below uses just 27 lines of code to create a simple login interface.

```

1
2 use LIMS::ArrayPipeLine;
3
4 my $pipeline = pipeline->new_guest('User_Login');
5 my $q = $pipeline->get_cgi;
6 my $base_url = $pipeline->base_url;
7 $pipeline->set_short_sidebar;
8 $pipeline->print_trl_header;
9
10 if ($q->param('logout')){
11     $pipeline->log_out;
12     print $q->p({-style=>'font-style: italic;'}, "You_have_been_logged_out");
13 } else {
14     print $q->start_form(-method=>"POST",
15         -action=>"http://$base_url/cgi-bin/lims.index.cgi"),
16     $q->hidden("web_id", $q->param('web_id')),
17     $q->p(' &nbsp; '),
18     $q->table( Tr( td({-style=>'text-align: right;'}, 'User_Name:_'),
19         td(textfield(-name=>'user_name', -size=>20))),
20         Tr( td({-style=>'text-align: right;'}, 'Password:_'),
21         td(password_field(-name=>'password', -size=>20)))),
22     $q->p(' &nbsp; '),
23     $q->submit(-name=>'Login');
24 }
25 $pipeline->finish;

```

2.3.3 Extending the ArrayPipeLine LIMS database for sample data

In total, eleven new tables were added to the ArrayPipeLine LIMS database to capture all the sample information a laboratory would create through the process of performing high throughput aCGH projects. A graphical representation of the

tables that were added can be seen in the entity-relationship diagram in Figure 2.1. As can be seen in Figure 2.1 by the number of relationships they each have, the three most important tables that were added were the `patient_info`, `specimen` and `tumour` tables. The `patient_info` table holds the details of a patient for which a sample has been recorded, the information in the table relates to clinical features of the patient such as the patient's age at diagnosis, the outcome of the disease and the length of time before the outcome. The specimen table is a non cancer specific sample table that allows a patient to have one or more normal samples such as a peripheral blood sample that might be used as a normal control in an aCGH experiment. If a patient's sample is a tumour, the specimen table can also record the additional information associated with tumour samples through a relationship with the tumour table. A tumour entry in the database records clinical information such as the stage and grade of the tumour. It can be seen in the figure that the external sample table `ext_sample` is not related to any of the other tables, but this important table allows the addition to the database of samples missing collection, specimen and tumour information, thus making downstream analysis pipelines available for these samples.

2.3.4 Extending the ArrayPipeLine LIMS database for aCGH experiment data

In order to be able to capture all the information created by an aCGH experiment required the addition of the most database tables of the three different data sources, eleven new tables in total. The entity-relationship diagram in Figure 2.2 represents all the new database tables and their relationships, and this clearly demonstrates how the database structure reflects the aCGH experimental workflow. The tables

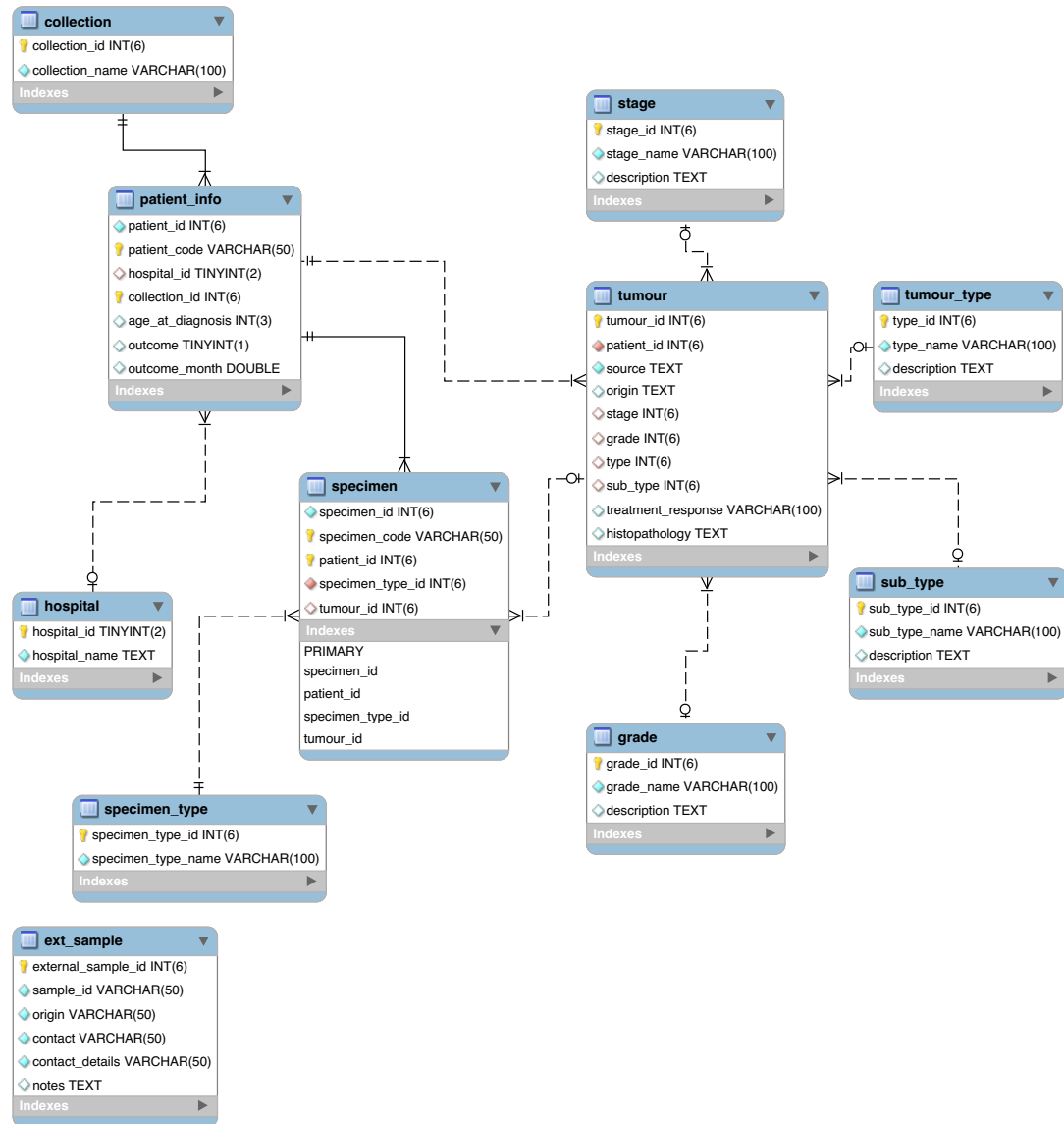


Figure 2.1: This figure shows an Entity Relationship Diagram (ERD) of the sample information portion of the ArrayPipeLine LIMS. Each box represents a new table in the database and each row in the box represents a column in the table with solid lines between boxes representing identifying relationships and dotted lines non identifying relationships. Primary key fields are shown as yellow, foreign key fields are shown as red and all other fields are shown as blue. Filled diamonds show fields in the table which are unable to take null values and empty diamonds show fields in the table which can take null values (figure created with MySQL workbench <http://wb.mysql.com/> using crow's foot notation)

represent the key experimental steps, while relationships between the tables both show the passage of data through the system and the order in which the steps take place.

The experimental process begins with fluorescent labelling of test and control samples. This step is recorded in the database by the **target** table which links a biological sample to a fluorochrome in the **fluorochrome** table. Additional information regarding the efficiency of the labelling reaction can also be stored in this table for quality control purposes. Following the process of labelling, the samples are then hybridised to a microarray; this is recorded in the database by the **cgh_experiment** table, which associates up to three fluorescently labelled samples with a microarray slide and a procedure identifier. The procedure identifier links to the **procedure_info** table, which records details of a laboratory procedure, including: a description of the procedure, who carried out the procedure and when it was carried out. All this information about the aCGH experiment is then given a unique experiment identifier generated by the table. After hybridisation, the microarray slide is scanned using a laser to produce images for each fluorescent label used in the experiment; the details of each scanned image are captured in the database in the **image** table. This table records where the image resides on the file system, the experiment identifier linking it to the samples used and details of how the image was generated including the software used and a number of values which relate to the settings used for the data collection.

The last step of the aCGH experimental process, modelled by the new database tables, is the image analysis of the scanned images, in which intensity values for each spot on the microarray are generated from the laser scanned images. The **image_analysis** table is central to this process, recording the location on the file system of the results file created by the image analysis, the image analysis

software applied to the data and a number of values relating to the quality of the scanned images used. As more than one image is used in the process of image analysis, the `processed_images` table was created to break the many to many relationship between the `image` and `image_analysis` tables, recording each of the image identifiers used in the image analysis.

There were two safeguards in place to prevent problems that would arise if any of the stored files were moved on the file system and their location and not also updated in the database. The first is the systematic naming used for all stored files, database identifiers were used to name each file that had been processed, in this way it would still be possible to match a file to the data being held in the database. In a situation where stored files were not only moved but also renamed it would be possible to restore the file system to a given time point in the previous month, by using a filesystem snapshot which are generated daily as part of the laboratories backup protocol.

We made the decision not only to store the location of the data file created by the image analysis, but also to store the spot intensity data in the database as well, with the two tables `scanarray_data` and `bluefuse_data` designed for this purpose. The choice to record the intensity data from the image analysis in the LIMS database was made for a number of reasons, including: the database data would act as an additional form of data backup in case the original files became corrupted; the database data would be faster to access programmatically; and to provide the ability to execute complex queries more easily. The `scanarray_data` and `bluefuse_data` tables record data from the intensity files created by the image analysis methods for each spot on the microarray. These intensity files contain a very large number of values for each spot. However only values that cannot be calculated are recorded in the database to reduce redundancy in the data. The reason for creating a table

for each image analysis method is that the Scanarray and Bluefuse software differ in approach, with each method generating a number of unique values. Rather than having a single data table containing multiple redundant values we chose to use individual tables for each method.

The unlinked table `automator_file` is concerned with the tracking and management of each aCGH scanned image and image analysis results file processed by the LIMS, providing details of any changes made to the file names and the location on the file system of every file.

2.3.5 Extending the ArrayPipeLine LIMS database for analysis pipeline data

The database design for recording information on the analysis steps applied to aCGH data involved the addition of seven new database tables to the original schema. The two key tables in this set of seven are the `data_analysis` and `processed_data` tables. The `data_analysis` table records the action of transforming a data set in some way and the `processed_data` table records the results of a data transformation. The `data_analysis` table propagates a unique identifier for every data transformation and to this identifier it associates the original scanned data source and the transformation which has taken place. The `processed_data` table records several values for each feature in the transformed data set which can include normalised \log_2 intensity ratios, segment level \log_2 ratios and aCGH call probability. However, not all the values need to be entered for a feature, which allows for the application of different analysis pipelines.

The remaining five tables are concerned with the recording of analysis pipelines; the `pipeline_info` table simply records the name and description of a unique

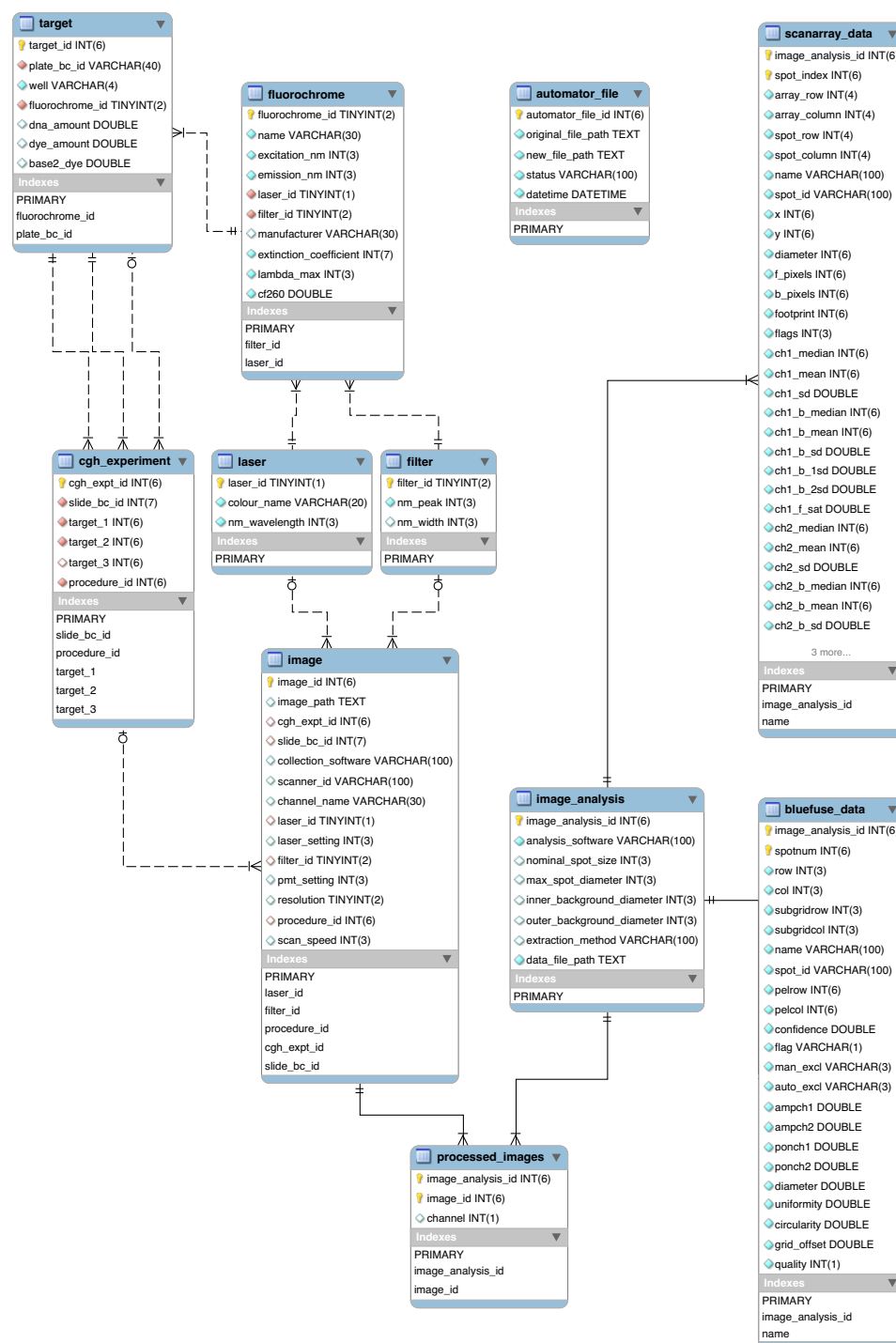


Figure 2.2: This figure shows an Entity Relationship Diagram (ERD) of the aCGH experiment portion of the ArrayPipeLine LIMS. Each box represents a new table in the database and each row in the box represents a column in the table with solid lines between boxes representing identifying relationships and dotted lines non identifying relationships. Primary key fields are shown as yellow, foreign key fields are shown as red and all other fields are shown as blue. Filled diamonds show fields in the table which are unable to take null values and empty diamonds show fields in the table which can take null values (figure created with MySQL workbench <http://wb.mysql.com/> using crow's foot notation)

analysis pipeline identifier which is generated by the table; the `analysis_pipeline` table records the order of analysis steps applied in a pipeline and associates these to an analysis pipeline identifier. The `analysis_info` table records each analysis step linking together an analysis name with a param set which is a record of the parameter settings used for the method, which are stored in the `param_set` table.

2.3.6 Extending the ArrayPipeLine GUI

Work carried out again in combination with my supervisor, lead to development of number of new views for the user interface, to support the addition and retrieval of the new data the LIMS database was now capable of supporting. I created interfaces for recording the details of biological samples in a sample collection figure 2.6. Added functionality for recording extra information associated with tumour samples was also created figure 2.4. Details of external samples which did not belong in any of the laboratories' collections could also be recorded in a separate interface I developed figure 2.7. I also developed an easy to use interface for recording details of the labelling protocol figure 2.5 which allowed users to detect problems with labelling reactions by calculating QC scores using measurements from a spectrophotometer. Super user functionality was also added for those users deemed responsible enough, administration pages allowed super users to create new sample collections as well as other higher functions figure 2.8.

The development of the new interfaces for user interaction in the ArrayPipeLine LIMS was made far simpler using the LIMS Perl modules described earlier in this section, since the modules enabled the generation of far simpler code that is easy to maintain. The LIMS objects also significantly increased the speed of development of new user interface web pages. As each web page is divided into a header, sidebar and footer, the HTML for each can be stored in a configuration file. There is a

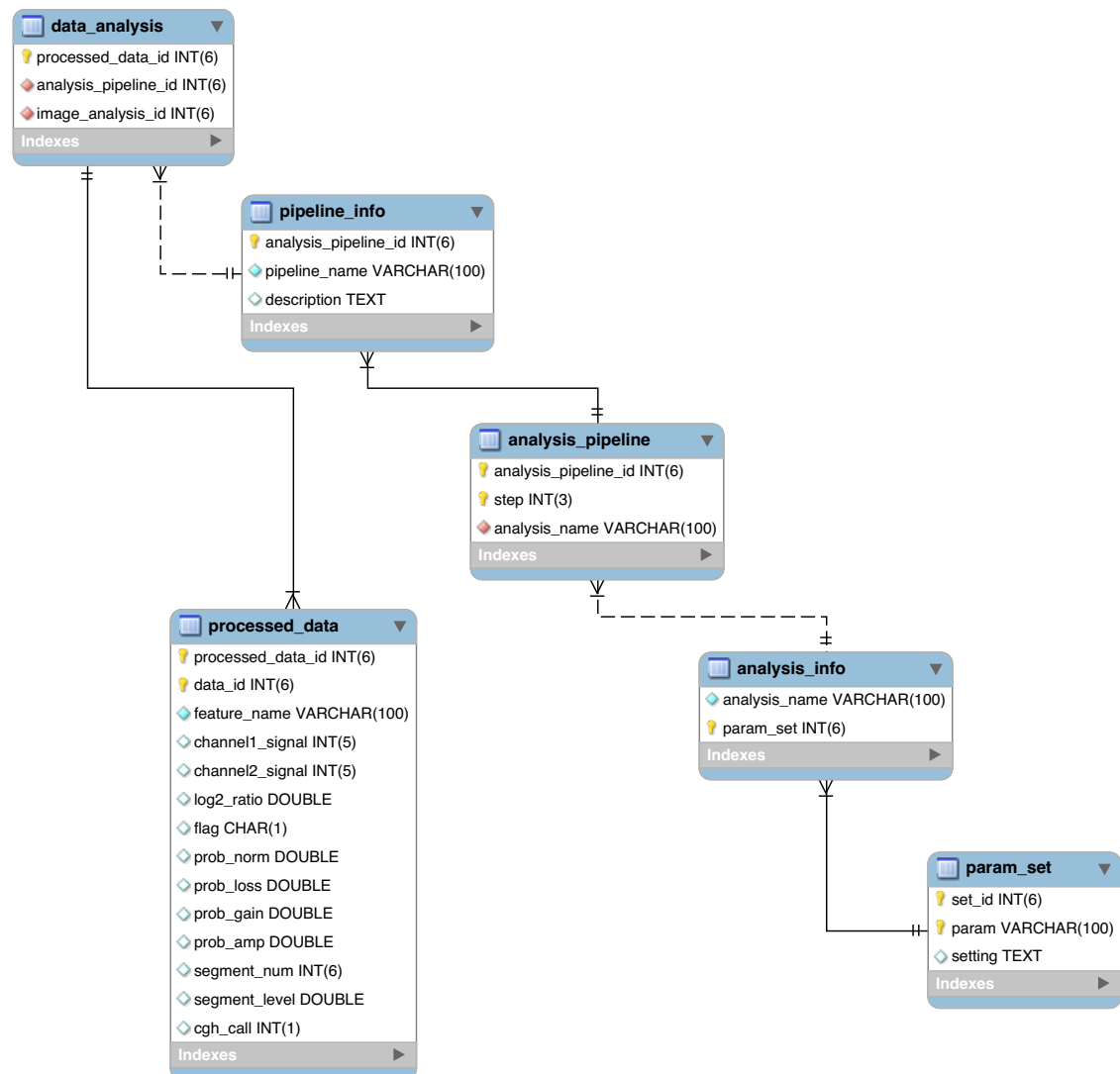
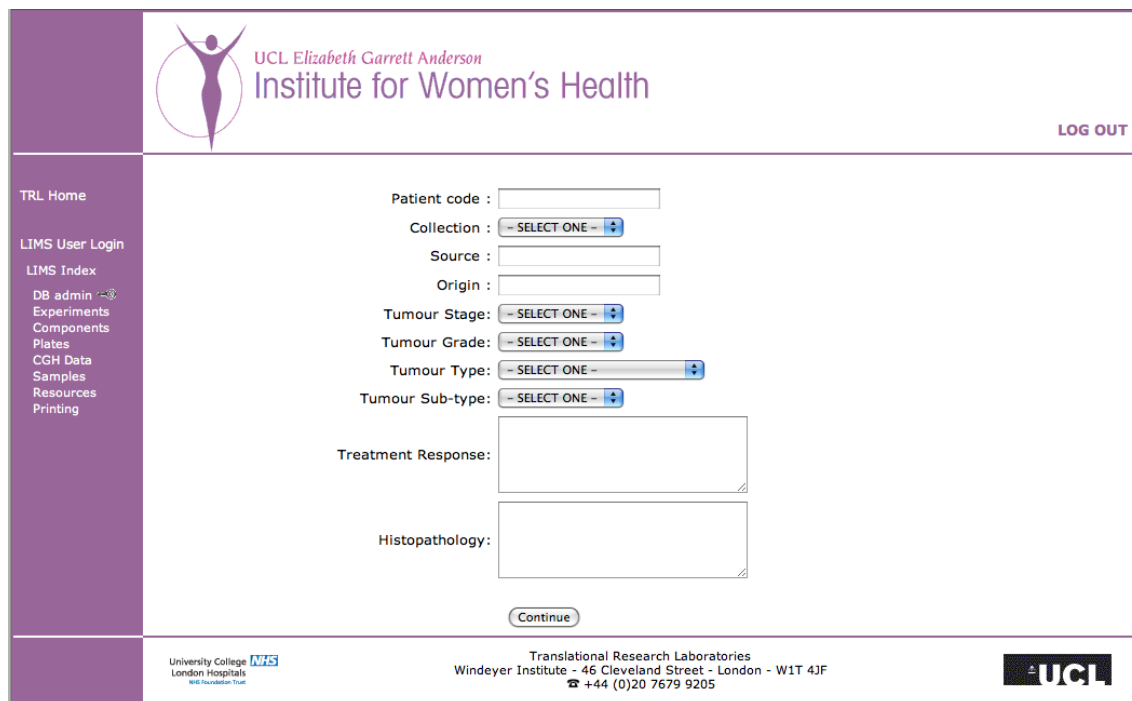
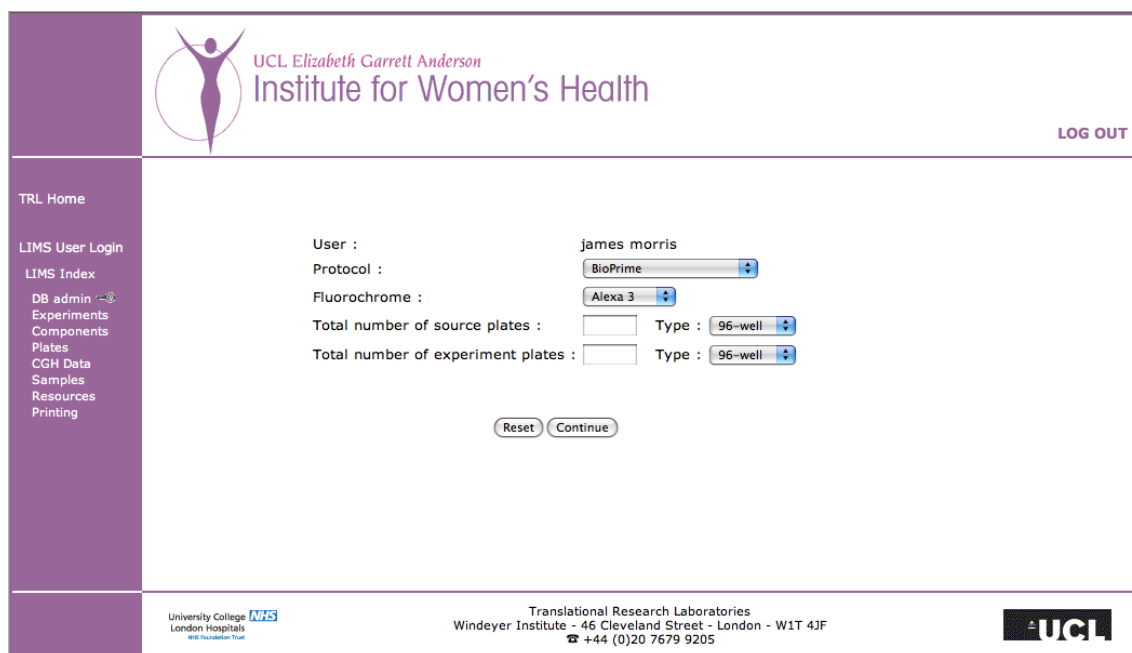


Figure 2.3: This figure shows an Entity Relationship Diagram (ERD) of the analysis pipeline portion of the ArrayPipeLine LIMS. Each box represents a new table in the database and each row in the box represents a column in the table with solid lines between boxes representing identifying relationships and dotted lines non identifying relationships. Primary key fields are shown as yellow, foreign key fields are shown as red and all other fields are shown as blue. Filled diamonds show fields in the table which are unable to take null values and empty diamonds show fields in the table which can take null values (figure created with MySQL workbench <http://wb.mysql.com/> using crow's foot notation)



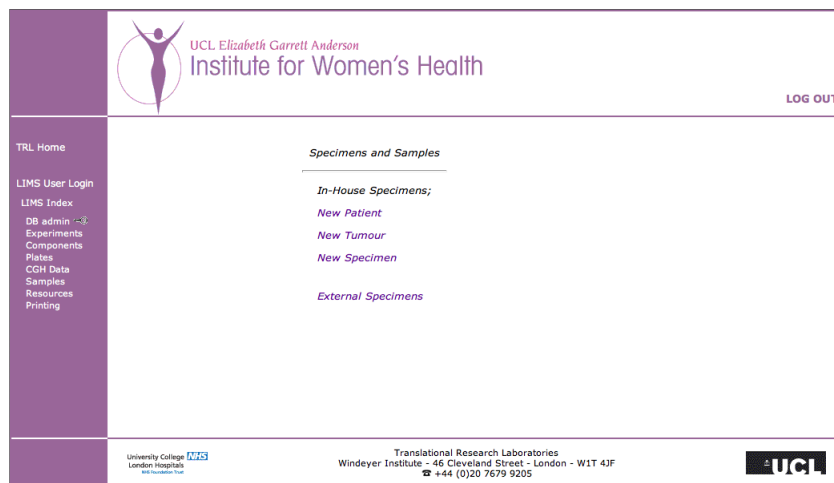
The screenshot shows the 'new tumour' interface of the ArrayPipeLine LIMS. The header includes the UCL Elizabeth Garrett Anderson Institute for Women's Health logo and a 'LOG OUT' link. A left sidebar contains navigation links: TRL Home, LIMS User Login, LIMS Index, DB admin, Experiments, Components, Plates, CGH Data, Samples, Resources, and Printing. The main form area contains the following fields: Patient code (text input), Collection (dropdown menu), Source (text input), Origin (text input), Tumour Stage (dropdown menu), Tumour Grade (dropdown menu), Tumour Type (dropdown menu), Tumour Sub-type (dropdown menu), Treatment Response (text area), and Histopathology (text area). A 'Continue' button is located at the bottom of the form. The footer displays the University College London Hospitals NHS logo, the Translational Research Laboratories address (Windeyer Institute - 46 Cleveland Street - London - W1T 4JF), and the UCL logo.

Figure 2.4: Screenshot of the ArrayPipeLine LIMS new tumour interface

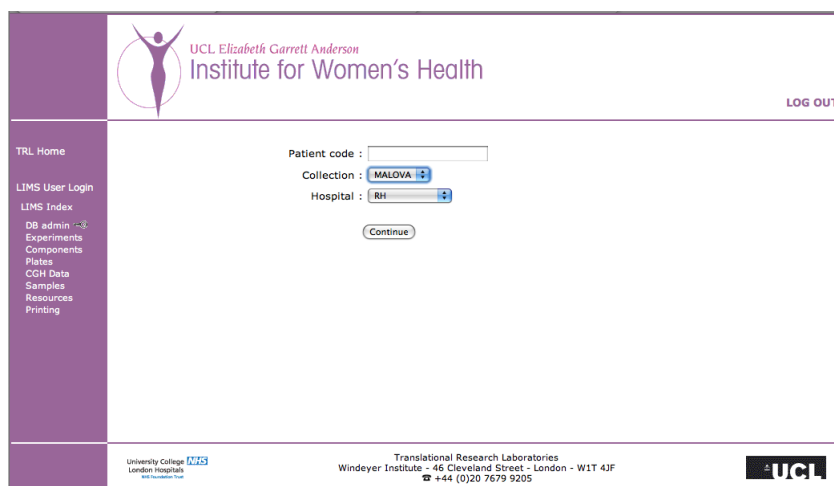


The screenshot shows the 'record labelling experiments' interface of the ArrayPipeLine LIMS. The header and sidebar are identical to Figure 2.4. The main form area contains the following fields: User (text input, value: james morris), Protocol (dropdown menu, value: BioPrime), Fluorochrome (dropdown menu, value: Alexa 3), Total number of source plates (text input) with Type (dropdown menu, value: 96-well), and Total number of experiment plates (text input) with Type (dropdown menu, value: 96-well). 'Reset' and 'Continue' buttons are located at the bottom of the form. The footer is identical to Figure 2.4.

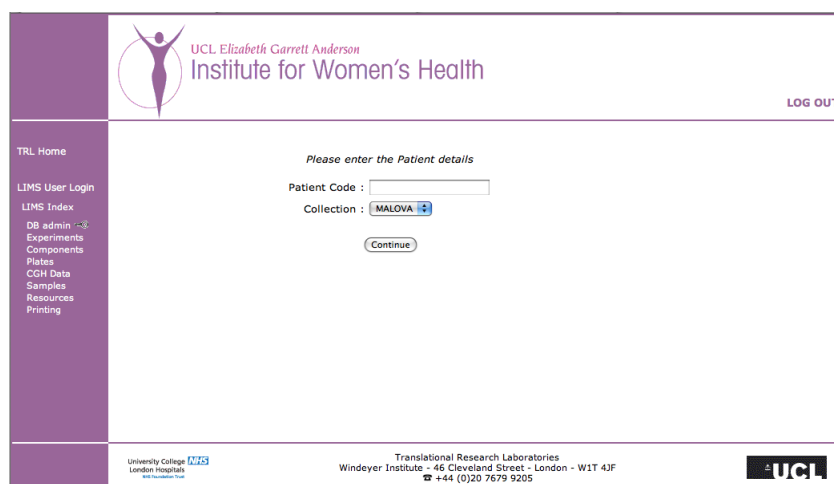
Figure 2.5: Screenshot of the ArrayPipeLine LIMS interface for recording labelling experiments



(a) Screenshot of the ArrayPipeLine LIMS samples index



(b) Screenshot of the ArrayPipeLine LIMS new patient interface



(c) Screenshot of the ArrayPipeLine LIMS new specimen interface

Figure 2.6: Screenshots of the ArrayPipeLine samples interface.

The screenshot shows the initial ArrayPipeLine LIMS interface. The header features the UCL Elizabeth Garrett Anderson Institute for Women's Health logo and a 'LOG OUT' link. The left sidebar contains a menu with items: TRL Home, LIMS User Login, LIMS Index, DB admin, Experiments, Components, Plates, CGH Data, Samples, Resources, and Printing. The main content area displays 'External sample(s) plate type : Tube' with a dropdown arrow, followed by 'How will you enter the sample info?' with radio buttons for 'manually' (selected) and 'file import'. A 'Continue' button is at the bottom. The footer includes University College London Hospitals, NPS, Translational Research Laboratories, and UCL logos.

(a) Screenshot of the initial ArrayPipeLine LIMS interface for recording an external sample

The screenshot shows the ArrayPipeLine LIMS interface for recording details of an external sample. The header and sidebar are identical to screenshot (a). The main content area contains several input fields: 'Sample ID :', 'Plate Barcode/ID :', 'Origin :', 'Contact Name :', 'Contact Details (telephone/email) :', and 'Additional Details :'. A 'Continue' button is at the bottom. The footer is identical to screenshot (a).

(b) Screenshot of the ArrayPipeLine LIMS interface for recording details of an external sample

Figure 2.7: Screenshots of the ArrayPipeLine external sample interface.

large number of lines of code behind these simple to use methods, which means that developers working on an interface for a LIMS are able to focus on the content of the interface they wish to build.

2.3.7 Continuous integration

One agile development practice successfully used during the development of the ArrayPipeLine LIMS was continuous integration. Evidence for the use of continuous integration can be seen in figure 2.9, which shows the gradual increase over time of lines of source code and source files in the project version control system, reflecting the constant addition of the new features at each minor release of software. A more traditional approach with a structured release pattern, would produce spikes of development activity close to each major release iteration. Using a continuous integration approach allowed us to quickly respond to any changes in the demands of the user, resulting in a system that delivered all the required functionality and no more. Contrast this result with the possible outcomes of a project working to a set of static demands or a project that does not formally recognise the need to interact with their users. In both of these cases software may be delivered that is lacking new features, or containing redundant features due to changes occurring during the development process.

2.4 Conclusions

The extension of the ArrayPipeLine LIMS successfully created an informatics solution for the TRL laboratory users, for the management and analysis of the associated data produced by the aCGH experiment workflow. The work to extend the LIMS also yielded extra functionality which made the LIMS capable

UCL Elizabeth Garrett Anderson
Institute for Women's Health

LOG OUT

TRL Home

LIMS User Login

LIMS Index

DB admin

Experiments

Components

Plates

CGH Data

Samples

Resources

Printing

Protocol Name : Test

Protocol Description :

Protocol File : [Choose File](#) no file selected

Protocol Type : [Other](#)

[Continue](#)

University College London Hospitals

Translational Research Laboratories
Windeyer Institute - 46 Cleveland Street - London - W1T 4JF
+44 (0)20 7679 9205

UCL

(a) Screenshot of the ArrayPipeLine LIMS protocols management interface

UCL Elizabeth Garrett Anderson
Institute for Women's Health

LOG OUT

TRL Home

LIMS User Login

LIMS Index

DB admin

Experiments

Components

Plates

CGH Data

Samples

Resources

Printing

Name :

Fluorochrome ID :

Excitation (nm) :

Emission (nm) :

Extinction coefficient ($\text{cm}^{-1} \text{M}^{-1}$) :

λ_{max} (nm) :

Laser ID : [0](#) [Click here to add a new laser](#)

Filter ID : [1](#) [Click here to add a new filter](#)

Manufacturer :

[Continue](#)

University College London Hospitals

Translational Research Laboratories
Windeyer Institute - 46 Cleveland Street - London - W1T 4JF
+44 (0)20 7679 9205

UCL

(b) Screenshot of the ArrayPipeLine LIMS interface for adding new fluorochemicals

UCL Elizabeth Garrett Anderson
Institute for Women's Health

LOG OUT

TRL Home

LIMS User Login

LIMS Index

DB admin

Experiments

Components

Plates

CGH Data

Samples

Resources

Printing

User Name :

Password :

Confirm Password :

Telephone :

email :

Location :

[Continue](#)

University College London Hospitals

Translational Research Laboratories
Windeyer Institute - 46 Cleveland Street - London - W1T 4JF
+44 (0)20 7679 9205

UCL

(c) Screenshot of the ArrayPipeLine LIMS users management interface

Figure 2.8: Screenshots of the ArrayPipeLine administration interface.

of supporting the aCGH profiling of large projects. This was important for the laboratory as the intention was to perform aCGH analysis on large sample sets numbering in the thousands.

The reason why the extension of the ArrayPipeLine LIMS was a success was because it provided users with the exact features they requested, at the same time the new features were delivered in a stable and robust system. The robustness of the LIMS means that it is able accurately to store all the experimental data arising from the aCGH workflow, including precise details of every step of the process such as details of the manipulation, the biological sample being manipulated, the batch of the reagents being used and which laboratory user performed the manipulation.

The success of the extension of the ArrayPipeLine LIMS can be predominately attributed to the agile techniques and software development best practices used in the development process. The first software development best practice that was applied to this project was implementation of a system that allowed us to effectively reuse new and old source code, for this, we applied object oriented programming techniques. The process of converting the original LIMS source code and all subsequent source code to use object oriented techniques was described in section 2.3.2 of the results. The result of using an object oriented code base was the creation of tools that better modelled the underlying system, these tools also allowed us to generate solutions using fewer lines of code resulting in less redundancy and fewer mistakes across the project.

The implementation of a Version Control System (VCS) to manage and track the LIMS source code was a relatively simple task which had very positive implications on the project. The time required for developers on the project to integrate VCS into their workflow was minimal, but the results of using a VCS far outweighed these small overheads. The VCS helped more easily to manage situations where

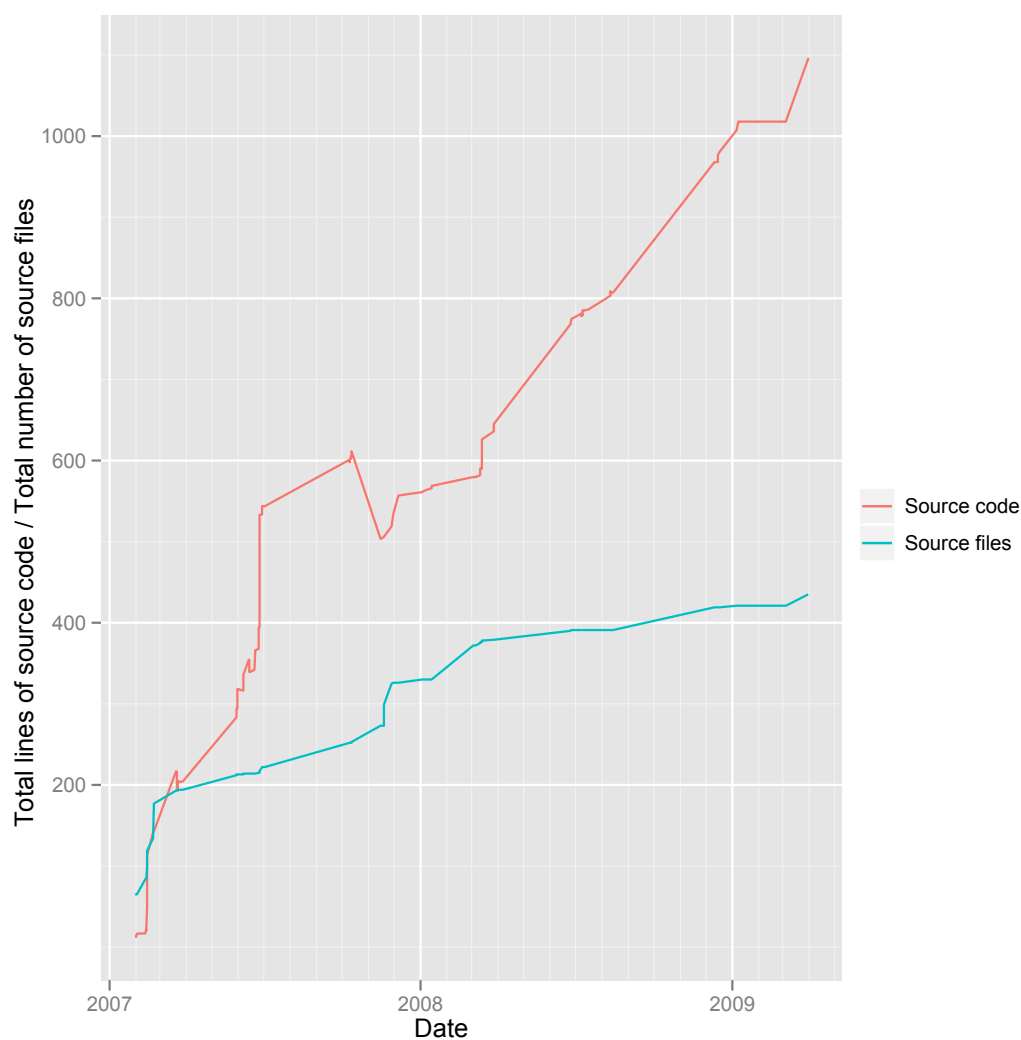


Figure 2.9: Continuous integration in the LIMS version control system. This figure shows the gradual accumulation of source files and lines of code over the development period of the ArrayPipeLine LIMS. The blue line represents the number of source files and the red line represents the number of tens of lines of source code plotted against date.

more than one developer was required to work on the same source code, enabling code integration without confusion or risk of code loss. The VCS was also invaluable in creating a robust system, as once a software bug was identified in the code the VCS could be used to track it down quickly. The workflow using the VCS after identifying a bug in the source code is as follows; retrieve the last working version from the repository; compare the broken version to the working version to identify differences; correct any of the differences that were causing the bug; finally test the new code and commit the new working version back to the repository.

An important decision in the design of the LIMS was the use of a relational database for storage of all data associated with the LIMS. While many systems utilise RDMS technology, the extensive use of foreign key constraints together with normalisation can be seen to complicate the database structure unnecessarily. However when a database is designed correctly and these features are fully implemented, as is the case with the ArrayPipeLine LIMS, databases can be an extremely effective tool for testing and validating the data imported and exported in a system and ensuring the logic of the system is maintained. The relationships between the tables added to the database and the use of effective type constraints on individual values in the tables enforce the integrity of the data in the database. For example the relationships in the database mean that data cannot be entered out of sequence such as entering the results of an experiment which does not exist in the database.

Extending the ArrayPipeLine LIMS using effective software development techniques including agile practices for bioinformatics has resulted in a thoroughly tested, robust system for recording and managing the data generated by microarray experiments specifically aCGH. The ArrayPipeLine LIMS therefore provides us with an unambiguous record of every aCGH experiment performed in our laboratory,

allowing us to perform meta-analysis across experiments with confidence in the data source. Without the use of the software development approaches described above, the LIMS would be likely to contain errors which we would not be able to detect, thus leading to inconsistent data which we would have less confidence in.

CHAPTER 3

An analysis pipeline for aCGH

3.1 Background

Microarray CGH experiments are complex multistep processes, introducing numerous opportunities for the incorporation of errors or biases into the experiment. The most common sources of errors and biases introduced into microarray experiments include:

- Sample handling errors created through human error, such as incorrect identification of samples used in an experiment.
- Results file handling errors, incorrectly associating the wrong scanned microarray image files or image analysis results file to an experiment.
- Fluorescent dye problems, such as inefficient labelling of a DNA sample or deterioration in probe quality.

- Hybridisation problems caused by uneven deposition of the hybridisation mixture on the microarray slide or inconsistent washing steps during the removal of excess hybridisation mixture before laser scanning, both of which can generate signal associated artefacts in the results.
- Laser scanning or image analysis problems caused by the use of suboptimal settings.

In order to detect the presence of these errors and biases and successfully remove their effect on microarray CGH data, it is necessary to carryout a number of data processing steps before interpreting the results of an experiment (Olson, 2006). Simple QC checks are able to detect basic errors, such as mistakes in sample handling. Simple visualisation techniques can be used to detect experiments with poor signal ranges or hybridisation or wash problems.

Once identified, the effects of many of these problems can be successfully corrected by the process of normalisation, a technique which aims to remove all biases from experimental data to leave only the biologically relevant signals (Yang *et al.*, 2002). Spatial and signal effects are the most common sources of bias in microarray CGH data (Neuvial *et al.*, 2006) and therefore there are specific normalisations methods that have been described for correcting these two different forms of bias (Neuvial *et al.*, 2006) and (Smyth, 2005). Signal normalisation aims to correct signal bias caused by technical limitations of the fluorescent dyes and laser scanners used. Spatial normalisation corrects experimental artefacts introduced by hybridisation and washing processes. Once normalisation is complete, differences between test and control samples should only be due to the biological system under study and not due to experimental effects.

Once the microarray data has been through QC and normalisation, a further two steps are required in order to produce biologically relevant results for researchers. Segmentation and copy number calling are both analytical methods which are exclusive to microarray CGH data and not required for other microarray application such as expression analysis (van de Wiel *et al.*, 2011). The process of segmentation aims to pinpoint the location of the breakpoints where copy number changes take place. Copy number calling methods aim to assign each microarray feature to one of three defined copy number states which are ‘gained’, ‘lost’, and ‘normal’. Some calling algorithms also have a fourth ‘amplified’ status for high level gains. This step provides researchers with clear biological meaning to the results of an experiment.

The extension of the ArrayPipeLine LIMS described in chapter 2 provided the laboratory with a well organised system for the tracking and storage of experimental data from microarray CGH experiments. Central to the LIMS system was an efficient relational database, capable of storing all relevant data generated by aCGH experiments. However a demand existed within the laboratory to develop a robust and flexible way to apply the informatics steps discussed in order to process the raw intensity data being stored in the LIMS database, to ensure data quality, generate meaningful biological results and aid with data visualisation. To achieve this goal required the ability to apply analysis methods sequentially forming an analysis pipeline, where a pipeline is simply a system by which the output of one process serves as the input of the next process and so on. In our case the initial input of the pipeline are the raw intensity values produced by a CGH microarray experiment, while the individual pipeline steps consist of separate analysis tasks, such as quality control filtering and normalisation.

The aim of this chapter of my thesis was therefore to fulfil the demands in the laboratory for a modular, automated analysis pipeline for the processing of

large amounts of experimental data. It was also necessary to create easy to use visualisation tools that would allow the user to interpret results generated by the analysis pipeline. The requirements of the system were for it to be flexible, in order to cope with rapid change in the analysis methods used, but at the same time consistently produce highly accurate results that could be developed in a timely manner. These requirements made the use of agile and extreme programming software development techniques highly appropriate for this project. Using agile (and more specifically, extreme programming) techniques while developing the analysis pipeline allowed us to test the hypothesis central to this thesis; that the best way to construct bioinformatics software is through the use of a formalised software development approach using agile techniques.

3.2 Implementation

3.2.1 Technical requirements of an analysis pipeline

At the time of designing the analysis pipeline it was evident from the literature that a great deal of effort had already been invested by researchers into developing statistical methods for the analysis of microarray experiments (Quackenbush, 2002). Owing to the fact that such a large amount of work had already been carried out developing these analytical methods, a decision was made that the analysis pipeline would be developed, where possible, using published methods for each of the analysis steps required. Using published analysis methods removed the need to develop new statistical methods, saving a great deal of time and resulting in more rapid assembly of the analysis pipeline. Using published analysis methods also provides the users of the analysis pipeline with a high level of quality assurance in the statistical methods

that comes from using methods that have already been peer reviewed.

A very large proportion of published microarray analysis methods elected to release their software as part of the Bioconductor project (Gentleman *et al.*, 2004) and new methods continue to do so. Since its first description in 2004, the Bioconductor project has been a very successful resource for the field of computational biology and bioinformatics. This success is due in no small part to the excellent development practices which were adopted by developers from the beginning of the project. The practices that are employed in the Bioconductor project include object-oriented programming techniques, modularisation of the source code and accurate and thorough documentation.

All of the methods we initially wanted to implement as part of the analysis pipeline were included in Bioconductor, enforcing our first design constraint upon the analysis pipeline; the statistical programming language R would need to be used for coding the analysis workflows. The statistical programming language R is a high-level interpreted language that was the chosen language for the Bioconductor project owing to a number of features which make it well suited for bioinformatics application development, including:

- The ability easily and quickly to prototype new computational methods as well as a great deal of support in the language for creating, testing, and distributing software in the form of ‘packages’.
- The Comprehensive R Archive Network (CRAN), contains several hundred open source packages addressing a wide range of statistical analysis and visualisation objectives.
- R has a well developed and tested set of functions and packages that provide access to different databases and to web resources.

- R includes a large number of built in statistical and numerical algorithms including machine learning algorithms.
- The increasing size of modern data sets makes the computational power required for analysis a very important consideration, therefore it is reassuring that R has also been the basis for pathbreaking research in parallel statistical computing a technique which can greatly improve the speed of computation.
- Finally perhaps the most important feature of R is its active user and developer communities.

A design decision was also made to the use of the programming language Perl for all non analytical tasks in the pipeline such as network tasks, pipeline management and the visualisation of analysis results. This decision was taken in order to maximise the interoperability between the existing ArrayPipeLine LIMS, which was fully developed in Perl, and the analysis pipeline to be developed.

3.2.2 Integrating Perl and R

As it became clear it would be necessary to use both R and Perl programming languages for the development of the analysis pipeline solution, an effort was made to investigate and test available methods capable of integrating these two languages in an attempt to simplify the pipeline development process. At the time this work was carried out there were two freely available solutions for integrating Perl and R, the RSPerl package (Lang, 2007) and a Perl module available from CPAN called Statistics::R (P. and Angly, 2011).

This package provides a bidirectional interface for calling R from Perl and Perl from R by embedding one interpreter (e.g. R) within the process of the other interpreter (e.g. Perl). The RSPerl package has the advantage of allowing users to call functions in the other language as if they were part of the local environment. Part of an example Perl script which uses RSPerl to perform spatial normalisation of aCGH data using the MANOR Bioconductor package is shown below to demonstrate how the RSPerl package may be used.

102

```

24 my ($arrayCol, $arrayRow, $spotCol, $spotRow) = $manor_data->array_design;
25 &R::call("setArrayDesign", $arrayCol, $arrayRow, $spotCol, $spotRow);
26 my ($aCol, $aRow, $aName, $aCh1_mean, $aCh1_bg_val, $aCh2_mean, $aCh2_bg_val) =
    $manor_data->array_values('mean', 'median');
27 &R::call("setArrayValues", $aCol, $aRow, $aName, $aCh1_mean, $aCh1_bg_val, $aCh2_mean,
    $aCh2_bg_val);
28 &R::eval("array <-<- list(_arrayValues=_arrayValues, _arrayDesign=_arrayDesign)");
29 &R::eval("class(array) <-<- \"arrayCGH\"");
30 my ($aClone_Names, $aChromosomes, $aPositions) = $manor_data->clone_values;
31 &R::call("setCloneValues", $aClone_Names, $aPositions, $aChromosomes);
32 &R::eval("array$aCloneValues <-<- cloneValues");
33 &R::eval("array$id.rep <-<- \"Name\"");
34 &R::eval("data(flags)");
35 &R::eval("rep.flag$args$rep.thr <-<- 0.1");
36 &R::eval("local.spatial.flag$args <-<- alist(var=_\"LogRatio\", _by.var=NULL, _nk_
    _=5, _prop=_0.25, _thr=_1.2, _beta=_1, _family=_\"gaussian\")");
37 &R::eval("SNR.flag$snr.thr <-<- 3");
38 &R::eval("SNR.flag$args$var.FG <-<- \"ch2_fmean\"");
39 &R::eval("SNR.flag$args$var.BG <-<- \"ch2_bg_val\"");
40 &R::eval("amplicon.flag$ampli.thr <-<- 2");
41 &R::eval("flag.list <-<- list(spatial=local.spatial.flag, _SNR=SNR.flag, _amplicon=
    amplicon.flag)");
42 &R::eval("xxarray.norm <-<- norm.arrayCGH(xxarray, _flag.list=flag.list, _FUN=median, _
    na.rm=TRUE)");
43 capture_R_output();
44 exit;
45
46 sub capture_R_output {
47     my $pid;
48     return if ($pid = open STDOUT, "|-");
49     die "cannot_fork:$_!" unless defined $pid;
50     open (FILE, ">/rout.txt");
51     while(<STDIN>){
52         print FILE $_;
53     }
54     close FILE;
55     exit;
56 }

```

Statistics::R

The Statistics::R module creates a bridge between Perl and R permitting the control of the R interpreter through Perl in different architectures and operating systems. The module is easy to setup, requiring only that the module is pointed towards the location of the local systems R binary file. The module also provides users with the ability to use the same R bridge for multiple Perl processes, avoiding the unnecessary computational overhead of creating and running multiple connections. Below is a Perl code snippet using the Statistics::R module to perform spatial normalisation of aCGH data using the MANOR Bioconductor package as in the previous code example:

```

1  use Statistics::R;
2
3  my $R = Statistics::R->new(r_bin=>'C:\Program Files\R\R-2.3.1\bin\Rterm.exe') ;
4  $R->startR ;
5
6  $R->send("require(MANOR)") ;
7
8  $R->send("spot.names<-c(\"Name\", \"ID\", \"F635.Mean\", \"F532.Mean\", \"B635.Mean\", \"B532.Mean\"); ac<-import(\" $input_file\", spot.names=spot.names, type=\"gpr\", sep=\"\t\")");
9
10 $R->send("pos<-read.table(\" $clonepos_file\", sep=\";\", as.is=TRUE); names(pos)<-c(\"Name\", \"Position\", \"Chromosome\"); ac$cloneValues<-pos; ac$id.rep<-\"Name\"");
11
12 $R->send("data(flags); local.spatial.flag$args<-alist(var=NULL, nk=$nk, prop=$prop, thr=$thr, beta=$beta, family=\"gaussian\"); rep.flag$args$rep.thr<-0.5; flag.list<-list(spatial=local.spatial.flag, spot=spot.corr.flag, ref.snr=ref.snr.flag, rep=rep.flag, unique=unique.flag)");
13
14 $R->send("ac.norm<-norm.arrayCGH(ac, flag.list=flag.list, FUN=median, na.rm=TRUE)");

```

```
15     _=TRUE); _ac.norm<-_sort.arrayCGH(ac.norm)");  
16     print $R->read ;  
17  
18     $R->stopR() ;
```

Limitations

After testing both the freely available Perl/R integration solutions, we identified a number of serious limitations in both of the solutions. The main limitation of the RSPerl package was the difficulty in the setup and maintenance of the system, caused by the way in which RSPerl very closely ties together the Perl and R software on system making them highly dependent on each other. This dependency makes it very difficult to update and maintain both pieces of software independently as is necessary to be able to utilise the latest features in each language. Which is an important consideration as the Perl core package currently has a monthly release cycle. The RSPerl package was also poorly supported on non Linux systems which was a problem as the laboratory IT infrastructure relied heavily on Mac OS X operating system.

The Statistics::R module avoided the setup and maintenance issues of RSPerl, but in doing so it provided a solution that was not true integration. Crucially Statistics::R does not allow data objects to be passed between languages, or functions to be called directly from the other language. The only communication that can be performed is using methods that pass and capture strings from each language. During testing of the Statistics::R module I detected problems with inconsistent results being returned, a situation which is unacceptable for an analysis pipeline. Due to the limitations we identified during our tests we felt that neither solution provided a simple and effective means for interacting with the R language from

within a Perl program to perform statistical analysis. We therefore decided not to pursue the use of either solution in the development of the analysis pipeline.

The chosen solution

The approach that was chosen for the analysis pipeline development was to proceed without integrating the Perl and R languages. Instead the pipeline would apply both languages separately, using the most appropriate language in each case, for instance; using R for statistical analysis tasks and using Perl for network operations and logging tasks. The analysis steps would be performed using R scripts that are launched and managed by means of Perl programs, while the consistency of the data would be maintained through transactions with the ArrayPipeLine relational database (accessed using the RMySQL package for R and the DBI module for Perl).

The approach that we chose does have some advantages. Using Perl and R in isolation in this way is very similar to the way many developers use Perl and HTML for the development of web services, in an approach known as templating. Templating systems for Perl are well established and are very popular for web development tasks. These systems allow developers to separate HTML and Perl source code allowing for clear separation of concerns, with Perl programmers able to concentrate on Perl programming and web developers able to concentrate on the HTML code. In the same way the analysis pipeline allows R programmers to write R scripts without worrying about knowledge of Perl, while allowing Perl programmers to write Perl code without knowledge of R.

3.2.3 User requirements of an analysis pipeline

In order to determine the required functionality for an aCGH analysis pipeline, we needed to consider the requirements of the two different users groups that would use

the analysis pipeline.

- Computational users will need to analyse large numbers of experiments quickly and easily in a robust framework, using the pipeline output for further downstream analyses such as combining information across multiple experiments to search for patterns in the data. The most important feature in an analysis pipeline for computational users is the format and availability of the output data.
- Laboratory users will need to quickly and easily assess the quality of an individual or small batch of experiments using simple to use visualisation tools. Laboratory users should not have to control the analysis pipeline, but instead need to be able to visualise processed data by simply providing the raw data files.

The two different user groups also had a number of feature requirements in common for the analysis pipeline:

- The ability to perform analysis in an automated or semi-automated fashion, making it easy to use and simple to analyse large numbers of experiments in one go.
- The need to have thorough error checking to ensure it provides high quality results.
- The need to provide a clear and simple to follow audit trail for all the analyses performed, ensuring the creation of highly reproducible data.

3.2.4 Pipeline construction

Considering both the technical and user demands that we identified, there were numerous ways in which a solution could be implemented. By far the simplest approach involves creating a standalone script that performs the required analysis from start to finish. This basic approach is very often selected in many laboratories as it has the major advantage of being fast to deliver which is an important factor in an academic research setting. The approach we chose for the analysis pipeline had a number of advantages over simple standalone analysis scripts, these include: the extensive use of logging to enable the quick identification and resolution of any problems encountered; source code tests to address concerns about robustness; automatic recording the settings of analyses maintaining reproducibility; the ability to apply exactly the same pipeline of analysis to every experiment in a large scale high throughput project; finally modular analysis units that can be combined easily in different ways to quickly create a variety of different pipelines for analysis.

3.2.5 Development techniques

In order to test the hypothesis that agile development techniques are ideally suited for the development of bioinformatics applications, the analysis pipeline was implemented using a number of agile practices in order to assess how they effected the development process. The agile practices that were chosen for this development task, along with some more general programming best practices, are as follows;

- A test first approach for the development of Perl analysis objects. The use of test first programming would create robust error free code that would produce accurate results. A test first approach generates a testing framework that allows for easy refactoring and addition of new features to the code base.

These tasks can be confidently performed with the knowledge that any errors introduced into the code base would be immediately identified and resolved.

- The on site customer practice was applied as the development team was located in the same open plan office space as the laboratory researchers who would use the software, allowing the frequent interaction of developers and customers.
- A practice of frequent software releases was adopted to ensure the project was progressing correctly by frequently giving the users something to test and comment on.
- Object oriented programming techniques were used to creating modular code that is easy to re-use allowing for rapid development.

3.3 Results

3.3.1 Analysis pipeline

Raw data processing

The first step of the analysis pipeline is raw data processing, which is concerned with the parsing and storage of new experiment files, represented in Figure 3.1 by a schematic diagram of the analysis pipelines raw data processing pipeline. This process is launched upon the addition of new files to an assigned data drop box on the laboratory server. In our own configuration of the analysis pipeline the raw data processing step is launched manually as we found that laboratory users would deposit data in batches making the constant monitoring of the drop box unnecessary. However constant monitoring of the data drop box would be trivial to implement through the use of a simple UNIX cron job or event handler. The raw data files

consist of a single image analysis output file containing raw intensity values along with a TIFF format image of the microarray slide for fluorochrome used in the experiment.

The first task undertaken by the data processing script is to parse the configuration file, from which the path to the data drop box and other environment variables are obtained. Following this, important metadata regarding the scanning process and the image analysis process is extracted and recorded in the database. Scanning parameters recorded in the information header of the TIFF images, such as laser and filter settings important for confirming the fluorochrome identity, are extracted using the Perl Image::ExifTool module; and details of the image analysis are extracted from the header of the raw intensity files, including the identities of the image files used in the analysis. The process of parsing these files and adding them to the database is aided through the use of a suite of microarray file handling Perl modules developed in the laboratory (<http://search.cpan.org/dist/Microarray/>). A total of five different QC checks are then made for each experiment to ensure data integrity, with the checks performed shown below:

- Has the identified raw data already been processed?
- Are the accompanying TIFF image files present in the data drop box, or are they already present in the ArrayPipeLine LIMS database?
- Do the microarray slide barcodes in the raw data file and image files match?
- Does the microarray slide used in the experiment exist in the ArrayPipeLine LIMS database?
- Are details of the experiment present in the ArrayPipeLine LIMS database?

After the new data from an experiment have passed these QC checks, all the relevant data are then parsed out of the files and entered into the ArrayPipeLine LIMS database. The data which are extracted and recorded in the database include the spot level intensity values for each channel as well as all additional values generated by the image analysis software that are not possible to calculate.

The final step of raw data processing is the archiving of the original data files. Without a copy of the original TIFF images it is impossible to re-perform image analysis on an experiment while the disposal would shorten the audit trail by one step possibly making it harder to investigate any problems that may be identified with the data later on. The absence of original files also makes it difficult to double check an important finding, possibly instead requiring researchers to repeat what are expensive experiments.

The TIFF image files and the raw intensity files are moved to a storage location that is set again in the local configuration file, which in the case of our laboratory is on protected RAID5 storage. Along with moving the files the system also systematically renames all files with their unique database ID in order to avoid possible errors and the details of each archived file is recorded in the ArrayPipeLine LIMS database.

As problems may occur at any step during data processing the analysis pipeline was built in a robust way to cope with any errors. Upon detecting a problem in anyone of the many QC checks the analysis pipeline generates an interpretable error message in the pipeline error log file and the data entered into the ArrayPipeLine LIMS database up to that point is removed and the database is rolled back to its state before the analysis pipeline was started, avoiding inconsistencies being introduced into the database. Removal of data from the database upon discovery of an error during the analysis pipeline is a measure made possible because all of

the database operations made by the analysis pipeline are made within one single database transaction. Database transactions are a single unit of work performed by the RDBMS, they are mainly used in this way to provide a recovery point following a failure, in order to maintaining consistency in the database.

Normalisation method selection

A two step normalisation procedure was selected for all aCGH data to minimise the impact of the two most common sources of bias in our experiments. The first most common bias in data was spatial effects introduced in the experimental process in hybridisation or washing stages. The next most common bias in the data was a signal bias across all the signal from a fluorescent dye, possibly caused by different affinities of the two dyes or by degradation of one of the dyes during the experiment. A handful of normalisation methods specifically for aCGH data were available, including: a stepwise framework designed to reduce intensity, spatial, plate and background bias (Khojasteh *et al.*, 2005) and a method called popLowess that normalised the effect imposed by copy number imbalances (Staaf *et al.*, 2007). However the stepwise framework approach was not made available as a software package and the popLowess method was tested but performed poorly, most likely because the type of bias it corrected was not effecting our data. For the first normalisation step to remove spatial bias in the data we instead chose MANOR (Neuvial *et al.*, 2006), chosen because at the time the analysis pipeline was developed it was the only spatial normalisation solution developed for aCGH data. Limma (Smyth, 2005) was chosen for the second step of the normalisation procedure as it implements a method for performing a loess normalisation, loess being a regression modelling method that has been shown to be effective in normalisation of signal bias in microarray data (Yang *et al.*, 2002). Implementation of both of these normalisation methods was

made simple as both were developed in R as part of the Bioconductor project.

Segmentation method selection

The solution chosen for the segmentation step in our analysis pipeline was DNACopy (Venkatraman and Olshen, 2007), this decision was based upon both published performance and on considering the ease of implementation. At the time the analysis pipeline was developed there had been two studies (Willenbrock and Fridlyand, 2005; Lai *et al.*, 2005) comparing the performance of a number of segmentation algorithms, and both of these studies found that the circular binary segmentation method DNACopy performed very well in terms of sensitivity and specificity with similar or superior results to all the other methods tested. The GLAD (Hupé *et al.*, 2004) segmentation method that also performed well in the two method reviews was also evaluated, however DNACopy performed better than GLAD in the detection of known aberrations from a well studied ovarian cancer cell line. The DNACopy algorithm like MANOR and Limma was also developed as an R package within the Bioconductor project which again fitted well with the design of the analysis pipeline framework.

Copy number calling method selection

CGHcall (van de Wiel *et al.*, 2007) was selected to perform copy number calling on the identified segments. The CGHcall algorithm was selected firstly because it achieves high calling accuracy for aCGH data, the algorithm is able to do this by combining the previously calculated segmentation results with a more biologically accurate model than other methods. The method allows for fluctuations in the levels of the different copy number calls by using random effects and combines the segmentation results with a mixture model to obtain the most likely classification

per segment rather than per individual clone. The output generated by CGHcall was also more attractive as a solution in our pipeline as it produces four copy number states to better model the underlying biology and each call is a probability not a 'hard call' so we are able to set our own thresholds for deciding on a copy number call.

Creating analysis pipelines

The system for creating analysis pipelines centres around launching a series of analysis scripts coded using R. The management of this process is controlled using a suite of Perl modules, with the individual elements of the analysis suite detailed below.

The `LIMS::ArrayPipeLine::ManorAnalysis` module manages the execution of an R script that uses the Bioconductor package MANOR (Neuvial *et al.*, 2006). The MANOR package performs spatial normalisation on the microarray raw channel intensity ratios, correcting or removing spots in regions of bias. An example of how simple it is to create a `ManorAnalysis` module is given below. The pipeline object used in the script contains a number of methods for interacting with the `ArrayPipeLine` LIMS database as well as local configuration settings.

```
1  use LIMS::ArrayPipeLine::ManorAnalysis;  
2  
3  my $oManor_analysis = manor_analysis->new(  
4      analysis => 'manor_analysis',  
5      analysis_id => 'MANOR_nk22',  
6      params => {  
7          nk=>22,  
8          beta=>1,  
9          thr=>1.2,  
10         prop=>0.25,  
11         family=>'gaussian'}  
12  );  
13  
14  $oManor_analysis->pipeline_object($pipeline);  
15  $oManor_analysis->run($db_id);
```

The LIMS::ArrayPipeLine::LowessAnalysis module manages the execution of a R script that uses the Bioconductor package limma to perform lowess normalisation of microarray data to remove any systemic bias introduced through the differences in the fluorescent dyes used. The limma package takes as input raw intensity values for each channel of the experiment and corrects these values if a bias is detected. An example similar to that given for the ManorAnalysis module is given below for the LowessAnalysis module.


```

1  use LIMS::ArrayPipeLine::LowessAnalysis;
2
3  my $oLowess_analysis = lowess_analysis->new(
4      analysis => 'lowess_analysis',
5      analysis_id => 'Lowess_Normalisation_(default)',
6      params => {
7          span=>0.3,
8          iterations=>4}
9  );
10
11 $oLowess_analysis->pipeline_object($pipeline);
12 $oLowess_analysis->run($db_id);

```

The LIMS::ArrayPipeLine::CGHcallAnalysis module manages the execution of an R script that uses the Bioconductor packages DNACopy (Venkatraman and Olshen, 2007) and CGHcall (van de Wiel *et al.*, 2007) for segmenting and calling aCGH data. The packages take as input data, the normalised log₂ intensity ratios which are used to produce biologically meaningful copy number called genomic segments. An example of how to create a CGHcallAnalysis module is given below.

```

1  use LIMS::ArrayPipeLine::CGHcallAnalysis;
2
3  my $oCGHcall_analysis = CGHcall_analysis->new(
4      analysis => 'cghcall_analysis',
5      analysis_id => 'CGHcall_1.5',
6      params => {sdundo=>1.5}
7  );
8
9  $oCGHcall_analysis->pipeline_object($pipeline);
10 $oCGHcall_analysis->run($db_id);

```

In order to follow the principals of object oriented programming, the suite contains the LIMS::ArrayPipeLine::PipelineStep module. This module is a parent

class for all the previously described modules. The module provides common methods to reduce redundancy in the code base and allow new analysis modules to be created very quickly and easily with the writing of just a handful of simple methods using the powerful object orientation concept of inheritance.

The most important module in the suite is the PipelineAnalysis module which manages the execution of an analysis pipeline. This module acts as a container for PipelineStep objects, launching each analysis step in sequence as well as recording the details of the pipeline in the ArrayPipeLine LIMS database. The PipelineAnalysis module can also be used to retrieve the details of a previously performed analysis pipeline which can then be used to repeat the specific analysis steps. Since the PipelineAnalysis module records the exact setting of the pipeline, multiple versions of the same analysis pipeline can be recorded and used, such as creating a low medium and high stringency QC filtering pipeline to produce a range of results from which the most appropriate may be chosen. An example of generating an analysis pipeline from settings stored in the ArrayPipeLine LIMS database using the PipelineAnalysis module is given below.

```
1  my $oPipeline_analysis = pipeline_analysis->new(  
2      pipeline_name => 'default_analysis_pipeline',  
3      data => ['exp_1', 'exp_2', 'exp_3']);  
4  
5      $oPipeline_analysis->set_pipeline_object($pipeline);  
6      $oPipeline_analysis->launch_pipeline;
```

Logging

At every stage of both the raw data processing and analysis pipeline, details of all important actions taken as well as any errors encountered are recorded in log files,

enabling users very quickly and easily to discover and diagnose problems encountered by the pipeline.

3.3.2 Visualisation of results

The visualisation tools developed for the analysis pipeline consist of object oriented Perl modules which are capable of generating QC and results plots utilising the Perl GD module and Thomas Boutell's GD image library (<http://www.libgd.org/>). The plots are generated rapidly on demand for the web user interface each time a user requests a plot. This design feature allows us to save a considerable amount of disk space that would be required if we had to save each plot. The visualisation tools are split into two separate Perl modules; one for QC plots (`Microarray::Image::QC_Plots`) and one for generating CGH plots(`Microarray::Image::CGH_Plot`).

The QC plot module `Microarray::Image::QC_Plots` is capable of producing four different ways of visualising the quality of an aCGH experiment:

The MA plot is used for the detection of systematic biases in microarray experiments that can be generated by differences in the hybridisation efficiencies of the fluorescent dyes used. For each spot on the microarray, the MA plot shows the log of the intensity ratio of the two channels (M) plotted against the log of the average channel intensity (A). The calculation used to derive M and A are shown in equations 3.1 and 3.2 respectively.

$$M = \log_2 \frac{CH1}{CH2} \quad (3.1)$$

$$A = \log_2 \frac{1}{2} CH1 \times CH2 \quad (3.2)$$

The MA plot of a good quality experiment will show a single cluster of points centred around a M value of 1 as can be seen in Figure 3.2(a). On a MA plot of a poor quality experiment, in which there is bias in the signal intensities, the distribution of points will move away from a M value of 1, as can be seen in Figure 3.2(b). An example of how easy it is to generate a MA plot using the `Microarray::Image::QC_Plots` module is shown below.

```
1  use Microarray::Image::QC_Plots;
2  use Microarray::File::Data;
3
4  my $oData_File = data_file->new($data_file);
5  my $oMA_Plot = ma_plot->new($oData_File);
6  my $ma_plot_png = $oMA_Plot->make_plot;
7
8  open (PLOT, '>ma_plot.png');
9  print PLOT $ma_plot_png;
10 close PLOT;
```

The intensity scatter plot is very similar to the MA plot as it is also a method for detecting intensity channel biases in the data. The intensity scatter plot displays the distribution of the channel one raw intensity data points against the distribution of the channel two raw intensity data points. A good quality experiment will produce a scatter plot with the slope of the line that passes through the distribution at one see Figure 3.3(a) as an example. A poor quality experiment with a biases in one channel will produce a scatter plot with the slop of the line that passes through the distribution greater or less than one as can be seen in Figure 3.3(b).Generating an intensity scatter plot using the `Microarray::Image::QC_Plots` module is just as simple as generating an MA plot.

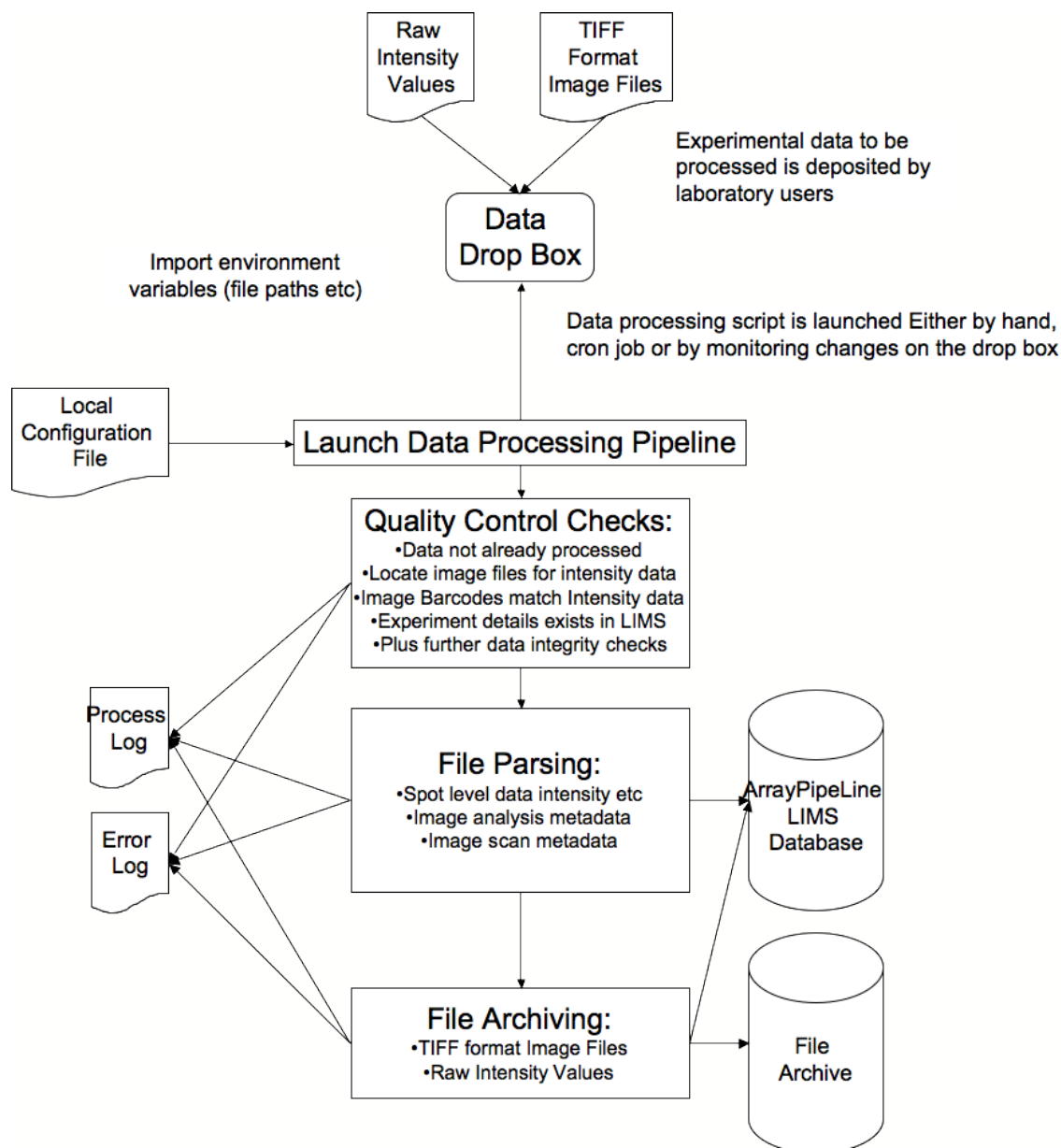
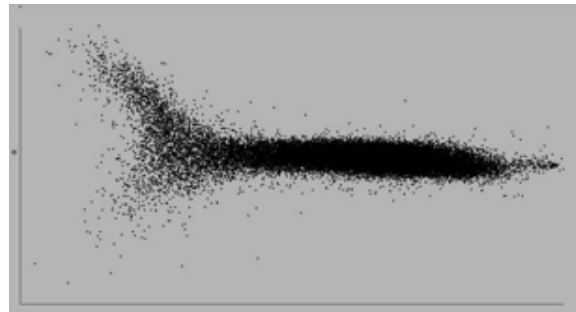


Figure 3.1: A schematic diagram showing the routes of data and the various steps that comprise the analysis pipelines raw data processing procedure. At the top of the figure raw data are first deposited in a designated data drop repository, next QC checks are made on the data and if QC is passed then the raw data are parsed and inserted into the ArrayPipeLine LIMS database. Finally in the last box of the figure the original raw files are archived. Additional files involved in the process can be seen on the left hand side of the figure, the local configuration file contains settings and parameters which are required for the process to run and the process and error log files record all the actions of data processing

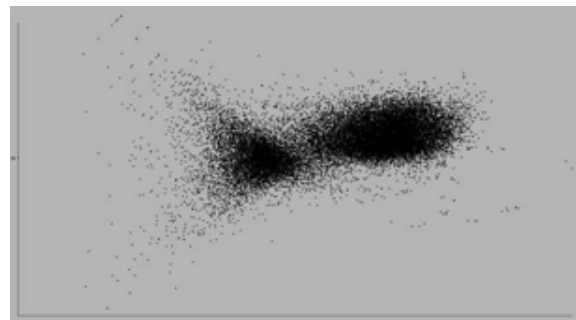
```
1  use Microarray::Image::QC_Plots;  
2  use Microarray::File::Data;  
3  
4  my $oData_File = data_file->new($data_file);  
5  my $oIntensity_Scatter_Plot = intensity_scatter->new($oData_File);  
6  my $intensity_scatter_plot_png = $oIntensity_Scatter_Plot->make_plot;  
7  
8  open (PLOT, '>intensity_scatter.png');  
9  print PLOT $intensity_scatter_plot_png;  
10 close PLOT;
```

A heatmap is a graphical representation of data where the values taken by a variable in a two-dimensional map are represented as colours. A microarray heatmap is constructed by plotting every spot on the microarray according to its X and Y coordinates on the microarray slide, and colouring each spot on a scale according to a relevant variable. Heatmaps are false colour representations of the microarray slide and as such they are very useful for detecting spacial effects within experiments. The QC_Plots module contains methods that allow users to generate two different forms of microarray heatmap plots: the \log_2 ratio heatmap and the channel intensity heatmap.

The \log_2 ratio heatmap function of the QC_Plots module can accept as input either raw or normalised values from an experiment. Individual spots on the plot are coloured according to the \log_2 of their signal intensity ratio; values greater than 1 are coloured red; values between 1 and 0 are scaled from red to yellow respectively; values between 0 and -1 are scaled from yellow to green; and values below -1 are coloured green. This plot will allow a researcher to quickly diagnose if a microarray experiment has been affected by any spatial bias. Spacial effects in microarray experiments are typically the result of problems with the hybridisation or washing steps of the experiment.



(a) MA plot of good quality data



(b) MA plot of poor quality data

Figure 3.2: MA plots produced by the QC_Plots module. Each point on the MA plot represents an individual spot on a microarray slide, they are plotted according to the average intensity across channels on the X axis and the channel intensity ratio on the Y axis. (a) the majority of the points in the MA plot lie on a horizontal line centred around 0 on the Y axis. This shows in a clear and simple way that the experimental data used to create the plot does not contain any signal biases and is therefore a good quality experiment. (b) in this plot the majority of the points do not lie on a horizontal line centred around 0, instead the data is split into two separate clouds with the larger cloud on the right moving away from 0. This indicates that a bias in signals from the two intensity channels exists, therefore quickly and easily demonstrating that the experimental data used to create the plot is poor quality.

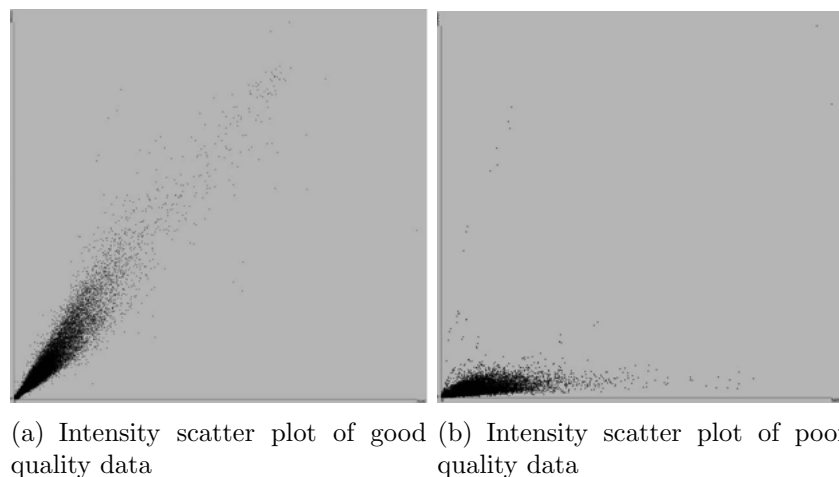


Figure 3.3: Intensity scatter plots generated by the QC_Plot module. Each point on the plots represents a spot on a microarray slide, plotted according to the first channel intensity on the X axis and the second channel intensity on the Y axis. (a) the data in this plot comes from a good quality experiment, this is clearly shown in the plot as the distribution of the points matches a slope with a gradient of one. (b) the data in this plot comes from a poor quality experiment. This can be detected easily as the distribution of points does not create a slope with a gradient of one, the cloud of points is pushed towards the X axis suggesting that either the first channel has failed to produce good signals or the second channel is producing far more signal than the first channel.

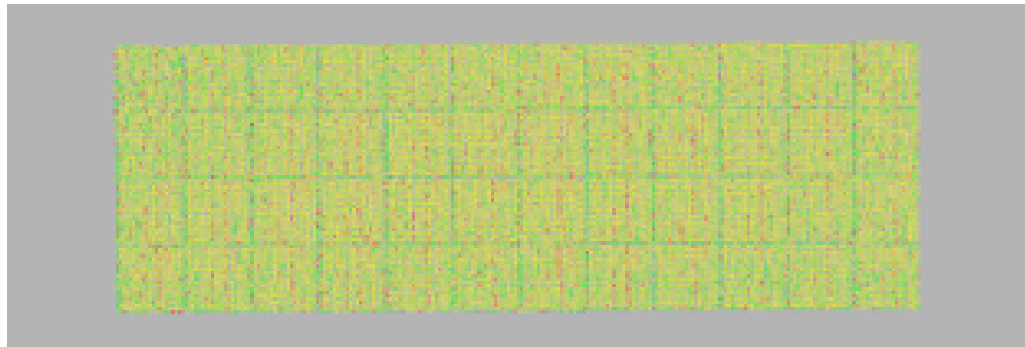
Single channel intensity heatmap plots allow users to investigate spatial effects further, as they provide the ability to identify the source of the bias. The intensity heatmap function in the QC_Plots module accepts raw intensity from an individual channel of the experiment, the points on the plot are coloured using a range which goes from black for an intensity of 0 to white for saturated microarray spots with the maximum intensity value that is possible to detect using a laser scanner.

The heatmaps in Figure 3.4(a) and 3.5(a) are examples from good quality experiments which show very little evidence of spatial effects as an even level of signal can be seen across the entire microarray slides. The heatmaps in Figure 3.4(b) and 3.5(b) are examples from poor quality experiments which clearly show a number of spatial effects, with irregular distribution of signal intensity across the microarray slides. The heatmap functions in the QC-plots module make it very easy to quickly

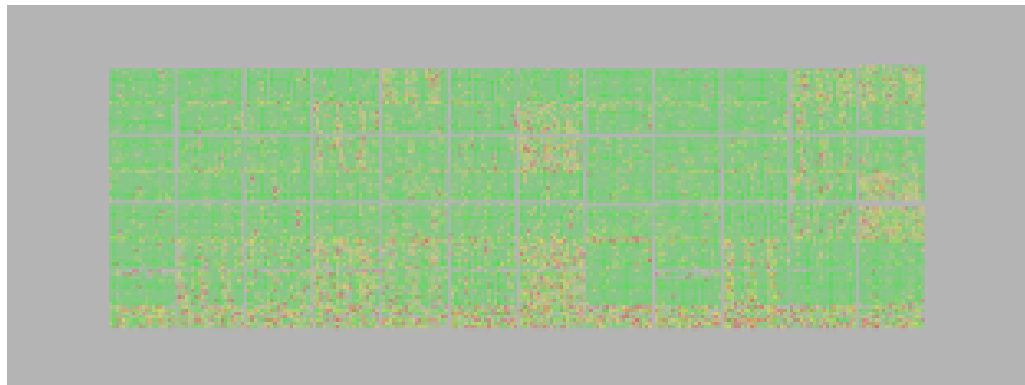
generate new heatmap plots, this is illustrated below with some example code for generating an intensity heatmap for the second channel of an experiment.

```
1  use Microarray::Image::QC_Plots;  
2  use Microarray::File::Data;  
3  
4  my $oData_File = data_file->new($data_file);  
5  my $oInt_Heatmap = intensity_heatmap->new($oData_File);  
6  my $int_heatmap_png = $oInt_Heatmap->make_plot(plot_channel=>2);  
7  
8  open (PLOT, '>int_heatmap.png');  
9  print PLOT $int_heatmap;  
10 close PLOT;
```

The other plotting module which is part of `Microarray::Image` is the `CGH_plot` module, which is a specialised module for creating a single aCGH plot. The aCGH plot displays the \log_2 ratio values of each feature on the microarray linearly according to its position along the genome, from chromosome 1 to the sex chromosomes X and Y. Using these plots it is possible to identify the shifts in ratio which equate to changes in copy number in the sample DNA. Figure 3.6(a) shows the an aCGH plot generated using the `CGH_plot` module, for each spot on the microarray, the \log_2 of the intensity ratio is plotted against the genomic location of the feature; segments of DNA which share the same copy number are shown using horizontal bars and the points and bars are coloured according to the copy number call assigned to the segment, copy number gain is coloured red, copy number loss coloured green and normal regions coloured yellow. CGH plots allow users to quickly and easily visualise copy number gains and can be used as another form of QC as the separation of regions of loss and gain from normal is a good indication of how well the experiment has worked.



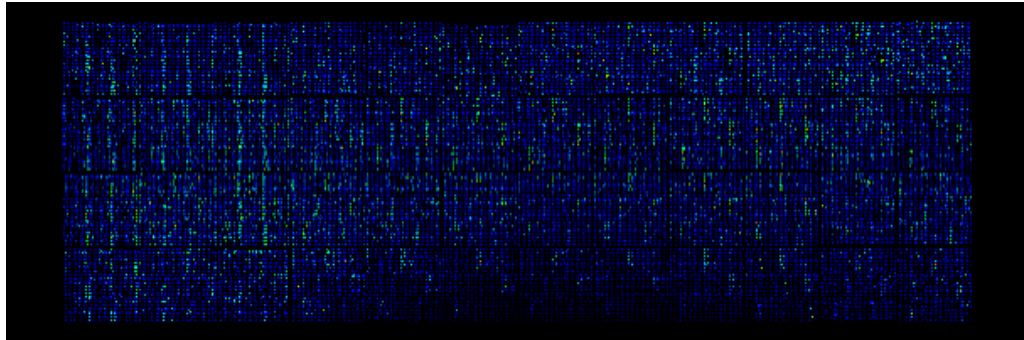
(a) Log_2 intensity ratio heatmap of good quality experiment



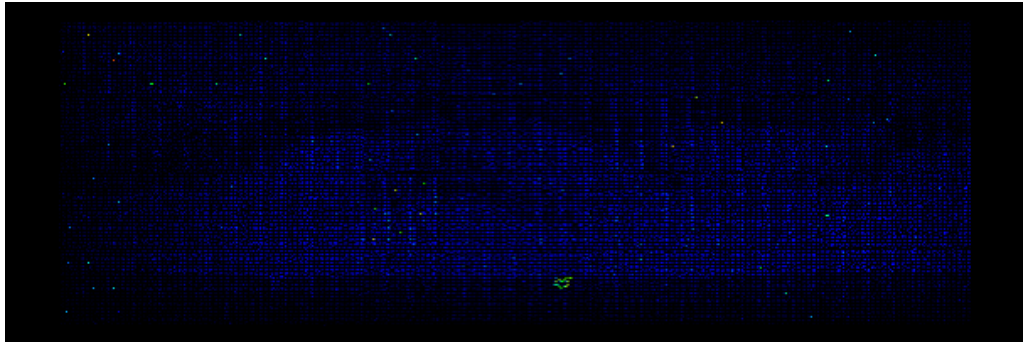
(b) Log_2 intensity ratio heatmap of poor quality experiment

Figure 3.4: Log_2 ratio heatmap plots generated using the QC_Plot module. Each spot on a microarray is plotted according to its X and Y coordinates on the slide, creating a false colour image of the microarray. The points on this heatmap are coloured according to log_2 intensity ratio using a colour scale that ranges from green (ratio of -1) to red (ratio of 1) with yellow (ratio 0) the intermediate. (a) this heatmap image clearly shows an even coverage of intensity values across the microarray with no concentrated regions of high or low intensity, such spatial effects can be introduced through problems in experimental stages such as hybridisation or washing. Thus the experiment that this data was taken from was of high quality. (b) the heatmap in this figure is an example from a poor quality experiment that has a spacial bias. The heatmap is clearly dominated by green points, this may have been caused by signal from the red channel being knocked out because of problems in the hybridisation process.

Because the features on the array are plotted according to genomic position, the `CGH_plot` module requires the location of each feature as well as the \log_2 ratio values. In order to allow researchers to concentrate on smaller more interesting regions it is possible to use the `CGH_plot` module to generate aCGH plots for individual chromosomes and an example of such a single chromosome plot is shown in 3.6(b).



(a) Single channel intensity heatmap of good quality experiment



(b) Single channel intensity heatmap of poor quality experiment

Figure 3.5: Single channel intensity heatmap plots generated using the QC_Plot module. Each spot on a microarray is plotted according to its X and Y coordinates on the slide, creating a false colour image of the microarray. The points on this heatmap are coloured by the intensity value from a single channel, ranging from black (intensity of 0) to white (saturated an intensity greater than the maximum intensity that can be measured by a laser scanner). This plot allows users if they have a detected a spatial bias using the log2 ratio heatmap to identify which channel in the experiment is causing the problem. (a) the heatmap shown here has bright spots across the entire microarray slide and does not display any regions of spatial biases as it is from a good channel in a good quality experiment. (b) the heatmap shown in this figure was created from data from a poor quality channel, this is obvious from the total lack of good quality strong signal spots that show as bright points on the heatmap. The heatmap also highlights a spatial bias in the data as the spots on the lower half of the array appear brighter than those spots on the top half.

```
1  use Microarray::Image::CGH_Plot;  
2  use Microarray::File::Data;  
3  use Microarray::File::Clone_Locns;  
4  
5  my $oData_File = data_file->new($data_file);  
6  my $oClone_File = clone_locn_file->new($clone_file);  
7  
8  my $oGenome_Image = genome_cgh_plot->new($oData_File,$oClone_File);  
9  my $oChrom_Image = cgh_plot->new($oData_File,$oClone_File);  
10  
11 my $genome_png = $oGenome_Image->make_plot;  
12 my $chrom_png = $oChrom_Image->make_plot(  
13     plot_chromosome=>1,  
14     scale=>100000);
```

3.3.3 Testing

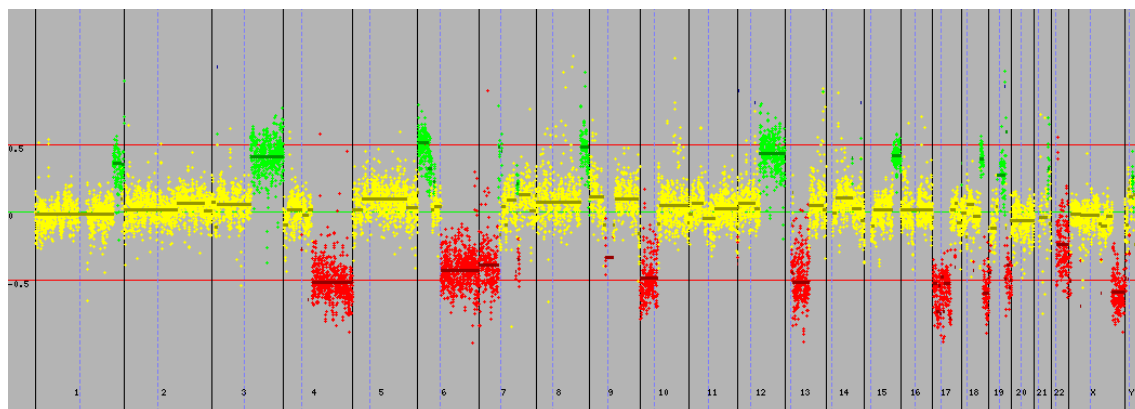
As a result of applying the XP practice of test driven development during this work, a test suite which supports all the code described was developed. The test suite contains tests capable of covering every required component of the analysis pipeline code base. The `process_data.t` test script was developed to test the raw data processing pipeline, run following any changes to the raw data processing pipeline, more than 250 individual test ensure no bugs have been indadvertedly introduced. The `pipeline_data.t` script runs more than 150 tests on the `LIMS::ArrayPipeLine::Pipeline_Data` module, which is an essential module for extracting data from the LIMS database, with the resulting objects used as input for running analysis pipelines modules. Finally the `pipeline_analysis.t` tests the functionality of the analysis pipeline modules using 181 individual tests. The tests were written in Perl using a number of specific testing modules including; `Test::Group`, `Test::Differences`, `Test::Deep` and `Test::More`. These modules

allowed for simple and rapid development of a testing framework, as they provide access to essential methods.

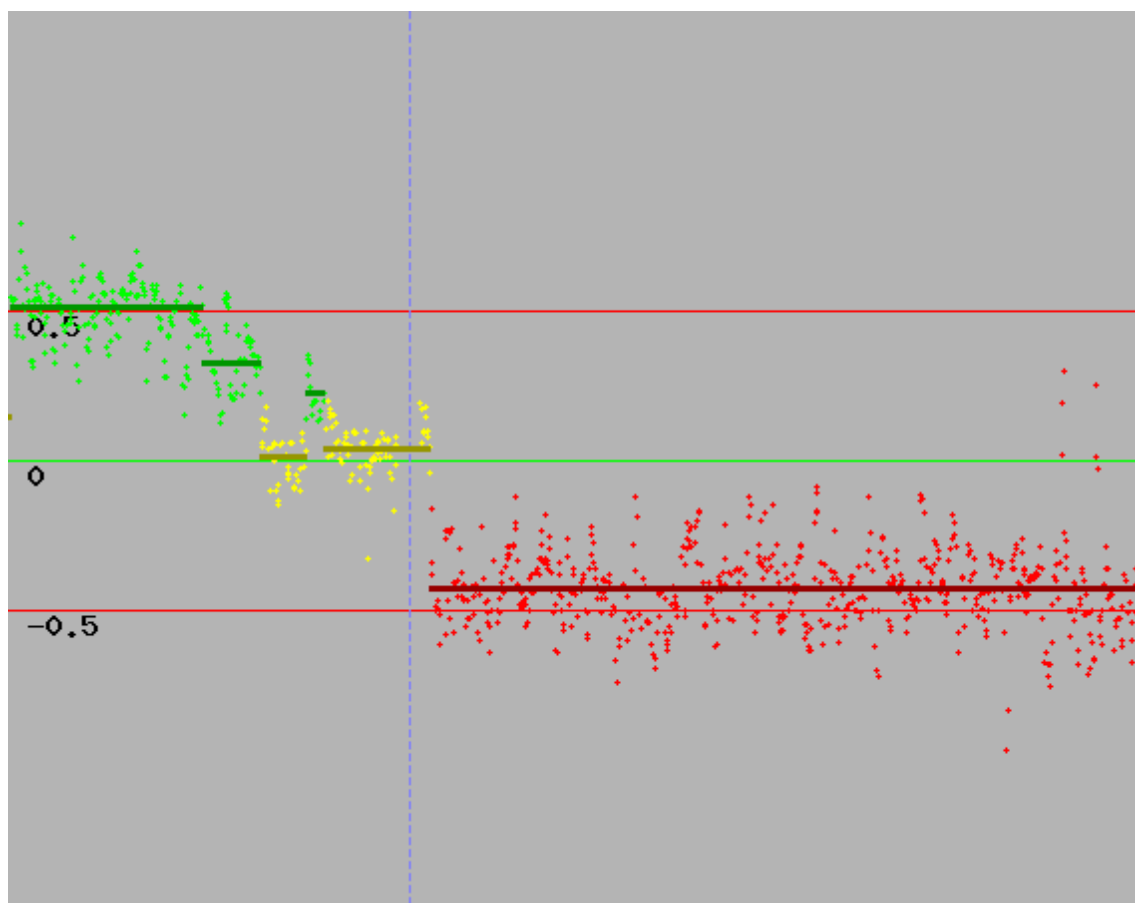
The test suite provided a number of important advantages in the development process; using a test first strategy meant that only the minimal amount of source code required to meet the requirements of a task were written. The developers were immediately aware that enough source code had been written once all the tests created were successfully passed by the new solution. The test suite also provided the developers with a ‘safety net’ that allowed source code to be refactored or improved with the reassurance that if the new code passed all the tests it would not cause any problems or generate any bugs.

3.3.4 Customer feedback

During the development described in this chapter we followed the agile practice of having onsite customers. Having customers working in the same office as the developers, in this case the customers were ArrayPipeLine LIMS users, allowed the developers to get rapid feedback on queries relating to the development of the LIMS. This practice was most useful during the development of the experiment visualisation tools. As an example of the impact of this practice, shown below are extracts from the version control log for the Image.pm module. The Image.pm module is a core module in the visualisation toolkit used for generating the QC and aCGH plots provided in the LIMS. The first line of each log entry states the revision number, the next line contains information on the date, revision author and the number of lines changed in the revision and the third line contains the developer message recording details of the changes made in the revision.



(a) Whole genome CGH plot of a primary ovarian tumour



(b) Single chromosome CGH plot of chromosome 6 from a primary ovarian tumour

Figure 3.6: CGH plots generated using the CGHPlot module. The aCGH plot displays the normalised log₂ intensity ratio of every feature present on a microarray according to its genomic location. Individual values from replicate spots present on the platform are combined to generate a single value for each feature. The X axis of the aCGH plot represents the genomic position of each feature, the Y axis of the plot is the normalised log₂ intensity ratio of the feature. Log₂ ratios approaching 0.5 are considered to be copy number gains, log₂ ratios approaching -0.5 are considered to be copy number losses and log₂ ratios around 0 are considered to have the same copy number as the control sample and are termed normal. The plots in this figure are also displaying the results of segmentation and copy number calling. Segments are depicted using horizontal bars which correspond to the segment average log₂ ratio, copy number calls are represented using the colouring system yellow are features with a copy number of 0 (normal), green features have a copy number of +1 (gain) and red are features with a copy number of -1 (losses). (a) whole genome aCGH plot from an ovarian tumour sample, vertical lines represent chromosome boundaries. (b) single chromosome aCGH plot of chromosome 6 from the previous whole genome aCGH plot. Using this zoomed in view it is possible to detect the small normal segment before the region of copy number gain that is not visible in the whole genome plot.

```
1 revision 1.8
2 date: 2007-06-25 18:08:25 +0100; author: james; state: Exp; lines: +390 -206;
3 printing whole and single chromosome plots
4
5 revision 1.7
6 date: 2007-06-21 16:21:08 +0100; author: james; state: Exp; lines: +253 -11;
7 moving average over a window of genomic distance
8
9 revision 1.5
10 date: 2007-06-07 16:44:31 +0100; author: james; state: Exp; lines: +10 -3;
11 added code to allow different clone sources for the cgh plot
12
13 revision 1.3
14 date: 2007-05-31 18:03:15 +0100; author: james; state: Exp; lines: +335 -98;
15 added log2 and intensity heatmap methods
16
17 revision 1.1
18 date: 2007-05-24 10:49:59 +0100; author: james; state: Exp;
19 Adding class for generating microarray plots
```

The first version of the Image.pm module revision 1.1 contained the basic functionality required for creating simple plots from microarray data. ArrayPipeLine LIMS users however requested the ability to generate more complex plots such as heatmaps for quality control processes. Therefore features were added in the third revision 1.3 of the module to allow users to generate an increased number of QC plots including heatmaps. The ability to load information on the genomic position of clones from multiple sources was later requested by users. The ability to load clone information from both files and databases was then delivered in the fifth revision 1.5 of the module. Users then required the ability to remove excess noise from aCGH plots in order to generate easier to interpret copy number profiles. This request was resolved by adding the ability to average the clone values in a sliding window revision 1.7. The final revision shown in this example was the result of users requiring the ability to look in greater detail at the copy number profiles of specific chromosomal

regions of the genome. To solve this the ability to generate aCGH plots for single chromosomes at higher resolution was added in revision 1.8. The version control log in this example shows a stepwise progression in functionality of the module, caused by the developers responding quickly to the feedback of feedback from the onsite customers of the ArrayPipeLine LIMS. This practice allowed the developers to generate software that were very close to the customers specification.

3.4 Conclusions

This chapter has described the construction of an analysis pipeline for aCGH experiments, from raw intensity values from aCGH experiments to meaningful biological results by assigning a copy number status of either loss, normal, gain or amplified to every feature on the microarray. The problem of effectively translating high throughput data into biologically meaningful results is common in laboratories using these techniques. As such, the solutions presented here and the findings from the development process using agile techniques is highly relevant. The process of converting raw data into biologically meaningful results is not a trivial informatics task, such an analysis process needed to be flexible, to allow the applied methods to be easily tuned in order to extract the maximum amount of high quality data possible. The system also needed to be well tested and capable of detecting any errors in the data to ensure all results being produced are reliable.

The idea of constructing modular systems for constructing reproducible data analysis pipeline is not a new one, at the time of development of our analysis pipeline solutions existed that had elements of the functionality we desired in our system including Taverna (Hull *et al.*, 2006), ICENI (Furmento *et al.*, 2002), Biopipe (Hoon *et al.*, 2003), Wildfire (Tang *et al.*, 2005) and Galaxy (Giardine *et al.*, 2005). Taverna and ICENI are both frameworks which use Web-Services for components, with the Taverna interface requiring the user to connect together output and input ports of components to build a workflow. Both of these tools are closely integrated with deployment of workflows over distributed computing resources or ‘grids’. Taverna also has an extensible architecture that allows integration of third party software tools. Biopipe is a workflow framework based on BioPerl which allows for execution of workflows across a single computing cluster. Wildfire is a framework that uses a

visual programming interface to allow users to compose programs into workflows that can be run over clusters or grids. In contrast to Taverna and ICENI, Wildfire works directly with program executables, rather than web-services. Galaxy differs from all the previous solutions, as it provides a web portal for integrating and interrogating existing datasets. Galaxy also allows users to filter and perform calculations on their queries as well as visualise them. An important feature of the Galaxy system is that every action the user makes is recorded and stored for reproducibility of results and analysis workflows.

All the workflow systems described above differ in their architecture as well as different features like workflow language, Primary data types, available resources, mechanism to add additional resources and extensibility. The workflow systems mentioned above can also differ in their ability to be implemented on parallel processors or grids (Tiwari and Sekhar, 2007). Although two of the solutions (Taverna and Galaxy) came very close to the specifications we required we chose to develop our own solution for executing and recording our analysis workflows due to a number of reasons. Unfortunately none of the frameworks natively supported both Perl and R, the two languages most favoured by our development team. Many of the solutions had additional complexity that was unnecessary for the purpose we desired, such as supporting grid technology which we did not have access to. Many of the solutions also had extensive user interfaces that would not be required with our analysis pipelines being managed and run by bioinformaticians with programming experience. The available resources of the frameworks mentioned was also biased towards sequence based analysis with little support for microarray data or analysis methods.

The bioinformatics solution we developed in our laboratory to meet our requirements centred around a suite of object oriented Perl modules which

managed the entire analysis pipeline, utilising statistical analysis methods from the Bioconductor project that are implemented in the statistical programming language R. In order to respond to demands from the laboratory based users of the analysis pipeline, object oriented Perl modules were also created that allowed the visualisation of the results and were incorporated into the ArrayPipeLine LIMS GUI. The analysis pipeline software that was developed makes it easy for users with basic knowledge of the Perl programming language to create different pipelines that can be applied to any experiment in the ArrayPipeLine LIMS database with very little effort, while also insulating users from the complex statistical Bioconductor packages implemented so far which can be difficult to master.

The result of using formalised software development approaches, including a number of agile development techniques, was the creation of two very successful software projects in the analysis pipeline suite of Perl modules and the visualisation modules. Evidence of the success of the analysis pipeline system comes from the fact that the system has been in constant use in our laboratory for more than three years, processing all of the laboratory's aCGH data, and remains in use today. The success of this software can be attributed mainly to the reliability of the software consistently to produce accurate results, which is the direct consequence of employing test-first programming in its development. The process of test-first programming creates software with fewer bugs and which is much easier to maintain, as only source code that is required for passing a test is written, cutting out anything unnecessary thus keeping the source code clean and easy to understand. The analysis pipeline modules have a test suite containing a large number of tests. We have not yet put the analysis pipeline Perl module on CPAN for use by other developers because of some dependancy the modules have with the ArrayPipeLine LIMS database. It would be more than possible with some development time to remove this dependancy by

allowing the user to define different data sources. This would enable other developers to utilise this code base for generating analysis pipelines.

The plots generated by the `Microarray::Image::QC_Plots` and `Microarray::Image::CGH_Plot` modules described in this chapter were made into an easy to use tool for the laboratory users, using a web based interface as part of the ArrayPipeline LIMS. Due to the rapid speed at which the two modules are capable of generating plots, the implementations in the ArrayPipeLine LIMS create plot images on the fly, when requested by the user. This approach has the significant advantage of saving large amounts of disk space that would otherwise be needed for caching all plots requested. These LIMS functions are used on a very regular basis by laboratory researchers for quality control purposes and visualisation of experimental results. The two Perl modules have also been made available on CPAN and are part of a wider set of tools created by our laboratory for managing the analysis of microarray experiments using Perl (Morris *et al.*, 2008b). The high level of user-satisfaction with the visualisation modules and associated LIMS tools can, like the analysis pipeline modules, be attributed to the application of agile development techniques. The visualisation tools of the ArrayPipeLine LIMS were developed using very high levels of customer feedback as we followed the principle of developers working in the same open office space as the customer.

In both the analysis pipeline and plotting projects, the use of the programming best practice of object oriented programming greatly increased the speed and quality of development. This was due mainly to the use of the object oriented concept of inheritance, where new classes are created as sub-classes of an existing super-class. The sub-class inherits the attributes and behaviours of the super-class reducing the generation of redundant code, resulting in fewer mistakes and easier to maintain, higher quality code.

An example of the use of object oriented inheritance in development of the code for this chapter comes from the creation of the three analysis pipeline modules (ManorAnalysis, lowessAnalysis and CGHcallAnalysis). The PipelineStep class was used as a super-class for the analysis sub-classes to inherit from and contains the majority of the methods required for performing an analysis step. The individual analysis modules required only a small number of new methods to be written in order for these modules to pass their tests. This was also the case for the code in the Microarray::Image modules where new plot classes such as ma_plot, intensity_scatter and intensity_heatmap were easily created using the Image module as a super-class. This module takes care of a large proportion of the common tasks required to generate a plot, meaning the development of the new plot classes only involved creating a handful of methods to handle the different forms of data.

A relatively simple way in which the work in this chapter could easily be taken further is through the development of a web interface to the analysis pipeline software. An easy to use interface that would allow users to build their own analysis pipelines and run them against any experimental data in the ArrayPipeLine LIMS database would be highly useful to the laboratory based users as this functionality is currently only available through the use of Perl scripts. Having the ability to select the specific steps to run on some experimental data as well as being able to specify all the analysis parameters would allow users that are not able to write Perl scripts to interact with the data in the ArrayPipeLine LIMS in a way they are not currently able to.

The idea of providing greater functionality for those users of the ArrayPipeLine LIMS analysis pipeline who are not literate in a scripting language such as Perl could be further expanded with the creation of an automated way to create new analysis modules for the analysis pipeline. Currently the only way to add a new analysis

module to the analysis pipeline is to write a (albeit simple) Perl module, restricting such users from adding new pipeline analysis steps that they might wish to evaluate.

The use of agile development techniques such as test first programming and customer involvement as described in this chapter has yielded highly successful software projects and serves as proof that these techniques can be used to very positive effect in a bioinformatics setting.

CHAPTER 4

The somatic genetic profile of an ovarian tumour is a strong determinant of prognosis

4.1 Background

DNA copy number profiling using aCGH was performed on 94 primary stage III/IV serous ovarian adenocarcinomas in the hope of identifying recurrent aberrations that may be linked to the mechanisms underlying the development of this cancer. Clinical information in the form of patient survival data was used in combination with the results of the aCGH profiling in an attempt to leverage even more information from the data.

A secondary aim in analysing the data from this project was the validation of the bioinformatic tools described in the previous two chapters. Both the ArrayPipeLine LIMS and the analysis pipeline Perl/R framework were developed to support large aCGH profiling projects, so the execution of this project using these tools would therefore serve as a good test of principle for the developed software.

This work involved the research and implementation of existing statistical approaches and techniques as well as the development of novel methods for the downstream analysis of aCGH data. In contrast to the single experiment analysis processes that have been discussed so far in this thesis, downstream analysis processes involve the combination of data and analysis across multiple aCGH experiments.

4.2 Materials and methods

4.2.1 Patient data

This study was performed on formalin-fixed, paraffin-embedded (FFPE) ovarian tumour samples from the MALOVA (MALignant OVArrian cancer) study, explained in more detail here (Høgdall *et al.*, 2003, 2004; Kjaerbye-Thygesen *et al.*, 2006). MALOVA is a multidisciplinary Danish study covering epidemiology (lifestyle factors), biochemistry and molecular biology with the purpose of identifying risk factors and prognostic factors for ovarian cancer. Preoperative blood samples as well as tumour tissue samples were obtained from most of the patients from the participant hospitals with a primary epithelial ovarian tumour. A total of 681 OC patients and 235 women with LMP were included in the MALOVA study (Høgdall *et al.*, 2007).

a population based collection of malignant ovarian cancer cases, containing 667 ovarian tumour sections accompanied by highly detailed epidemiological data. All the samples selected for investigation were FIGO grade III or IV with a patient age less than 70 years old, FIGO stages were obtained from clinical records and were reviewed by two gynaecologists, both specialised in ovarian cancer (Høgdall *et al.*,

2007). In total 94 samples were profiled using the ‘Mermaid’ aCGH microarray (Dafou *et al.*, 2009). The samples were divided into two data sets based on patient survival; the first data set contained samples from patients with a time to outcome greater than 60 months, where the study outcome was patient death. The second data set contained samples from patients with a time to outcome that was less than 24 months. A training set of samples was formed through carefully matching by means of age(\pm 2 years), tumour stage and tumour grade, pairs of samples from the two different prognosis data sets. This approach yielded a training set of 80 well matched samples from 40 good and 40 poor surviving cases. The remaining 14 unmatched samples formed a validation data set to be used for the accuracy testing of any classification models created by the downstream analysis.

All of the following experimental steps, up to scanning the hybridised microarray slides (data analysis section 4.2.5), were all carried out in our laboratory by the tumour profiling groups research assistant Tanya Lebi.

4.2.2 Sample preparation

Formalin fixed paraffin embedded (FFPE) tumour sections were reviewed by a pathologist, and regions containing greater than 80% neoplastic cells were identified. Tumour sections were needle microdissected from a single 10x field ensuring that 80% or more of the dissected cells were neoplastic in character. The DNA of the dissected cells was extracted by proteinase K digestion. Genomic DNA was extracted from peripheral blood lymphocytes (PBL) by Whatman.

4.2.3 The Mermaid CGH microarray

The ‘Mermaid’ array (Dafou *et al.*, 2009) is the in house manufactured aCGH platform, a whole genome tiling-path microarray that has spotted on its surface 32,450 BAC clones from the human genome high-resolution BAC re-arrayed clone set (Krzywinski *et al.*, 2004) supplied by the BACPAC Resource Centre (BPRC), part of the Children’s Hospital Oakland Research Institute (CHORI) in Oakland, California, USA. The set provides coverage of the human genome sequence in excess of 99%. The tiling-path configuration provides us with an effective genomic resolution for detection of copy number gains and losses of less than 0.5 Mbp. BAC DNA preparations were amplified by ‘rolling-circle amplification’ using the Phi29 polymerase (GE Healthcare) and purified using MultiScreen PCR384 filter plates (Millipore), before being denatured and diluted in a DMSO-based printing buffer (Genetix). The entire clone set is printed onto UltraGAPS microarray slides (Corning) at a feature diameter of approximately $110\mu\text{m}$, using a QArray printing robot (Genetix) and 50m silicon printing pins (Parallel Synthesis). NCBI Human Genome build 35 (HG17) BAC clone locations for the array features derived from BAC end-sequence and fingerprint analysis were provided by the BPRC and converted to build 36 (HG18) locations using the UCSC lift-over tool (Hinrichs *et al.*, 2006).

4.2.4 Microarray hybridisation

The patient’s tumour DNA was used as the test sample and the reference sample in the experiment was the patient’s PBL DNA where available, otherwise a commercial pooled female genomic DNA sample (Promega Life Sciences) was used. The test and reference samples were both amplified by whole genome amplification using the

WGA-II kit (Sigma). The samples were then differentially labelled using Alexa-3 (reference) or Alexa-5 (test) dyes using the BioPrime Total kit (Invitrogen). Non human and repetitive DNA sequence was then removed by prehybridisation with Cot1 DNA (Invitrogen). The labelled samples were finally hybridised to the microarray for 16-18 hours.

4.2.5 Data analysis

Slides were scanned using a ScanArray Express laser scanner (Perkin Elmer) at $5\mu\text{m}$ resolution and raw fluorescence data were extracted from scan images using BlueFuse software (BlueGnome, Cambridge, UK). The raw data were then imported and processed using our own bespoke data analysis pipeline, which utilises a number of R packages. First, a two stage normalisation procedure is performed, which involves firstly the detection and normalisation or removal of any local or global spatial effects using MANOR (Neuvial *et al.*, 2006); and secondly the data are \log_2 -ratio normalised to remove any dye bias using an implementation of the loess algorithm (Smyth, 2004) in LIMMA (Smyth, 2005). The data are then segmented into regions of discrete copy number by circular binary segmentation using DNACopy (Venkatraman and Olshen, 2007), and DNA copy number status for each BAC clone was determined using CGHcall (van de Wiel *et al.*, 2007).

4.2.6 Expert supervised filtering

After all the raw experimental data had been processed 20 samples were removed from the validation set. Reducing this set from 80 to 60 samples, 30 good and 30 poor. The excluded experiments were selected through manual curation of the aCGH profiles by an aCGH expert. Profiles displaying either of the following characteristics

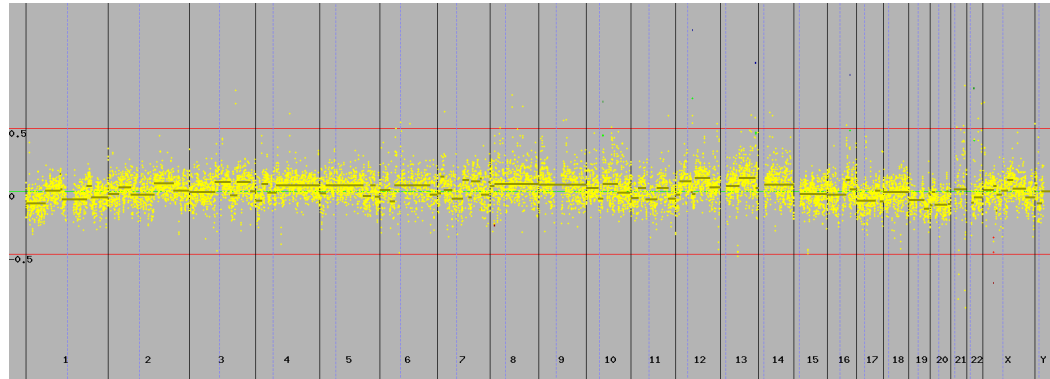
were excluded:

- The profile displayed minimal genomic instability producing what was termed a ‘flat profile’, an example of which can be seen in the first panel of figure 4.1.
- The profile exhibited a high degree of variation in neighbouring BACs in the experiment producing what was termed a ‘wide spread profile’, an example of which can be seen in the second panel of figure 4.1.

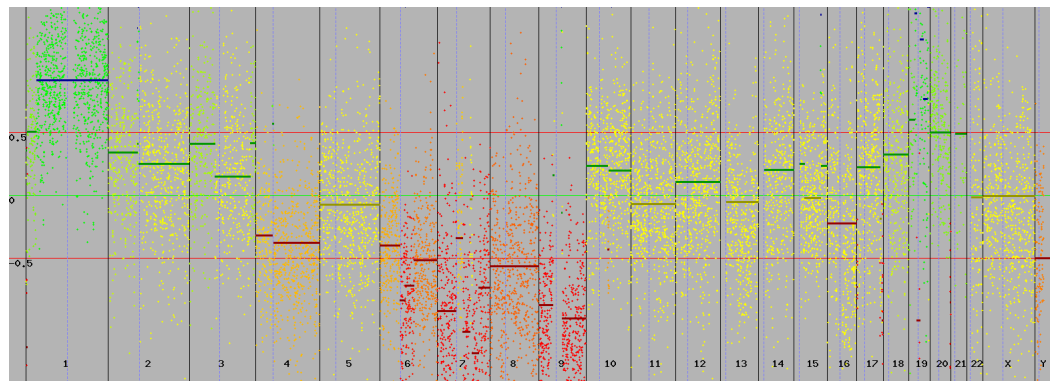
4.2.7 Statistical analysis

Recurrent region identification

Identification of recurrent regions of aberration was carried out across three different sample sets. The first data set comprised all 60 tumour samples in order to identify common aberrations. The remaining two sample sets contained the 30 good survival patient samples and the 30 poor survival patient samples, enabling us to identify aberrations unique to the different classes. Recurrent regions of loss or gain present across the different sample sets were identified using a statistical approach implemented in the R package KCSmart (Klijn *et al.*, 2008). The KCSmart package was chosen to over the other available methods discussed in the introduction. With no published comparison of these methods available they were difficult to compare, as all the methods looked at the problem in slightly different ways. We decided to instead consider more practical factors in selecting the best method such as availability of the software, software requirements, ease of installation, input data format and type of data output. The KCSmart package best met all the criteria we defined for the selection of a recurrent aberration detection solution. The criteria were as follows:



(a) Profile displaying minimal genomic instability.



(b) Profile exhibiting a high degree of variation in neighbouring BACs.

Figure 4.1: Two example aCGH profiles excluded from further analysis after expert supervised filtering. The aCGH plot displays the normalized \log_2 intensity ratio of every feature present on a microarray according to its genomic location. The X axis of the aCGH plot represents the genomic position of each feature, the Y axis of the plot is the normalised \log_2 intensity ratio of the feature. The plots in this figure are also displaying the results of segmentation and copy number calling. Segments are depicted using horizontal bars which correspond to the segment average \log_2 ratio, copy number calls are represented using the colouring system yellow are features with a copy number of 0 (normal), green features have a copy number of +1 (gain) and red are features with a copy number of -1 (losses).

- The software should generate a statistical confidence measure for the regions it identifies.
- The software should be implemented in a framework supported by the ArrayPipeLine LIMS.
- The software should be freely available and open source.
- The software should also be actively supported.

The input data for this analysis was first extracted from the ArrayPipeLine database as normalised \log_2 ratio values for each feature on the ‘Mermaid’ array for each experiment. These data was then filtered to remove features from the analysis which possessed values for less than 50% of the experiment in the given data set. For experiments which passed this filtering step, but which had missing values, an attempt was made to impute missing values to increase the density of data. This follows the methodology used in the KCSmart paper in the analysis of a set of breast cancer data (Klijn *et al.*, 2008). The imputation algorithm applied simply identified, where possible, one upstream and one downstream feature based on genomic location. The average \log_2 ratios from these two neighbouring features for the experiment in question was then assigned as the missing value. Each one of the three sample sets was analysed in the same way using KCSmart, three kernel smoothed estimates (KSE) (feature density = 0.5Mb) were generated using small (width = 2Mb), medium (width = 8Mb) and large (width = 24Mb) kernels and for each KSE a significance threshold was generated using a permutation approach (2000 permutations, $\alpha = 0.01$). We took advantage of the ability of the KCSmart algorithm to treat chromosome arms separately by providing telomere locations for each chromosome from the Homo Sapiens, Ensembl v49 release.

Recurrent region feature selection

We applied a simple algorithm to the significant recurrent regions identified by KCSmart in order to eliminate those regions which did not show a clear difference in occurrence between the good survival and poor survival datasets. For each recurrent region the algorithm calculated the average KCscore of all the BAC features contained in the region. The KCscore is a value calculated by the KCSmart algorithm, which represents a measure of the combined \log_2 ratio across all the experiments in the data set. If a significant recurrent region was identified from the good survival data set then the algorithm calculates the average KCscore of all the BAC features contained in the same region from the poor survival data set, or *vice versa*. The percentage difference of these two values is then calculated and if this difference is below a set threshold the region is rejected as being similarly recurrent in both good and poor groups: if the difference is above the threshold, then the region is selected as a feature for classification as it is differentially recurrent between the two groups under investigation.

Unique breakpoint identification

In an effort to reduce the complexity of the data set, we produced an alternative feature set comprised of every unique breakpoint in the project. This was achieved by looking across all experiments to find unique breakpoints to use as features. With all the segments identified by DNACopy for each experiment held in a database table it was simple to retrieve a list of distinct breakpoint values thus creating a list of unique minimal segments for the experiments in the project. The idea is based on combining the information in a segment to reduce the impact of non biological signals. Using this approach we were also able significantly to reduce the complexity

of the data.

Support Vector Machine analysis

Support Vector Machines (SVMs) were built in R using the `e1071` (Evgenia Dimitriadou and Andreas Weingess Hornik, 2011) library to classify the survival of patients based on data from the identified differential occurring recurrent regions of copy number gain and loss. The following list details the SVM options and settings that the `e1071` library requires to train a SVM:

- **kernel** the `e1071` library contains four kernel types:
 - linear $\mathbf{u}^\top \mathbf{v}$
 - polynomial $\gamma(\mathbf{u}^\top \mathbf{v} + c_0)^d$
 - radial basis $\exp\{-\gamma|\mathbf{u} - \mathbf{v}|^2\}$
 - sigmoid $\tanh\{\gamma\mathbf{u}^\top \mathbf{v} + c_0\}$
- **type** SVMs can be used as a classification machine, as a regression machine, or for novelty detection. The `e1071` library supports these different modes of operation using the `type` parameter and the following options:
 - C-classification
 - nu-classification
 - one-classification (for novelty detection)
 - eps-regression
 - nu-regression
- **fitted** boolean value indicating whether the fitted values should be computed and included in the model or not.

- **scale** logical vector indicating the variables to be scaled. If scale is of length 1, the value is recycled as many times as needed. Per default, data are scaled internally (both x and y variables) to zero mean and unit variance. The centre and scale values are returned and used for later predictions.
- **cost** cost of constraints violation it is the C-constant of the regularization term in the Lagrange formulation.
- **tolerance** tolerance of termination criterion.
- **epsilon** epsilon in the insensitive-loss function.
- **shrinking** option whether to use the shrinking-heuristics.

All the work shown in this chapter used the following parameter and setting when using the e1071 library: **kernel** = linear **type** = C-classification **fitted** = default: TRUE **scale** = TRUE **cost** = default: 1 **tolerance** = default: 0.001 **epsilon** = default: 0.1 **shrinking** = default: TRUE

To select the best combination of recurrent features to use for the classification, we carried out an analysis to assay the performance of every possible combination. For each combination of features, aCGH data from those chromosome regions was provided to an SVM for classification of survival. The accuracy which is measured as the percentage of correctly classified samples, achieved by each SVM was measured using a leave-one-out cross-validation approach. Classification performance was compared using either DNA copy number status calls provided by the CGHcall package, or individual \log_2 ratio data for each BAC.

All the possible feature combinations were generated by a function in the gtools library (cran.r project.org, 2012). The performance in terms of accuracy for each combination was recorded in a database table, making it easy to retrieve the best

performing combinations. Initial combination analysis looked to find the best performing features from a set of more than 1 million combinations of the 20 best ranked features. Performing the generation of over 1 million SVMs took over 5 days of computing time, thus to improve this performance we modified our code taking advantage of the Simple Network Of Workstations (snow) R package (Tierney *et al.*, 2012) allowing us to run the analysis in parallel making use of the 16 processor cores contained in our server. The parallelization of the analysis resulted in the analysis running more than 5 times quicker than the original approach, thus the required computing time for over 1 million different combinations was reduced to less than 1 day. The implementation of parallelization was very useful for the users of the downstream analysis pipeline, as it made it far more feasible to perform the SVM analysis described in this chapter each time new data was added to the ArrayPipeLine database. This was important because the generation of new experimental data in the laboratory was limited by some pieces of equipment, this subsequently resulted in aCGH results from new samples being created in batches of around ten samples. Having the ability to run the downstream analysis pipeline in a timely manner enabled the users to carefully monitor the effect new data had on the analysis and quickly detect any problems with data quality. The successful parallelization of this analysis process was also a useful proof of principle, that showed the value of this relatively simple computational technique in performing analysis tasks quicker without the requirement for more or higher performance computers.

Classifier validation

After combination analysis had yielded accuracy values for every possible combination of the selected features, it is a trivial task to select the best performing combinations from the database using a simple SQL statement. Once defined, the

best performing combinations were tested using an independent set of samples to validate the results and to show the classification model is not over fitted to the training data. For each of the best performing combinations a SVM model is built again using the full training set (n=60), this model is then used to predict the class of 14 test samples. In order to assess the predictive performance of each model an accuracy (A) value is calculated using equation 4.1 along with the model specificity (Sp) or true negative rate using equation 4.2, sensitivity (Sn) or true positive rate using equation 4.3 and Matthews Correlation Coefficient (MCC) which takes into account true and false positives and false negatives using equation 4.4. Where true positives denote poor prognosis patients correctly identified, true negatives denote good prognosis patients correctly identified, false positives denote poor prognosis patients incorrectly identified and false negatives denote good prognosis patients incorrectly identified.

$$A = \frac{TP + TN}{TP + FP + TN + FN} \quad (4.1)$$

$$Sp = \frac{TN}{TN + FP} \quad (4.2)$$

$$Sn = \frac{TP}{TP + FN} \quad (4.3)$$

$$MCC = \frac{(TP * TN) - (FN * FP)}{\sqrt{(TP + FN) * (TN + FP) * (TP + FP) * (TN + FN)}} \quad (4.4)$$

where TP, TN, FP and FN denotes the number of true positives, true negatives, false positives and false negatives, respectively.

Random forests

Random forests were constructed, in order to evaluate their accuracy in classifying the survival of patients, based on recurrent regions of copy number gain and loss. Comparing their performance to the SVMs produced by the e1071 library we would be able to select the best method to use for identifying the most informative regions. For constructing random forests we employed the R package `randomForest` (Breiman *et al.*, 2012). This implementation is based on the original Fortran code authored by Leo Breiman, the inventor of random forests. We applied random forests with parameter configurations for the values of `ntree` = 5000 (number of trees to build), `mtryFactor` = 0.5, 1, 2 (a multiplicative factor of the default value of `mtry` parameter denoting the number of features to be randomly selected for each tree), and `nodesize` = 1 (minimal size of the terminal nodes of the trees in a random forest), these parameters being consistent with the recommendations of the software documentation.

Gene prioritisation

In order to identify possible oncogenes, or tumour suppressor genes, whose copy number might impact survival, genes within the regions used for classification were analysed using CGprio (Furney *et al.*, 2008). The CGprio tool generates a probability of how likely a gene is an oncogene or tumour suppressor, using a prediction method based on a hidden Markov model built using gene and protein properties that are likely to contribute to a gene's potentiality to be oncogenic.

To investigate regions of interest using the CGprio tool, the Ensembl gene ID of every gene present in the region was retrieved using the Ensembl release v49 biomart tool (Kasprzyk *et al.*, 2004). The list of Ensembl gene IDs was then entered into

the CGprio web based interface. For each gene, CGprio returns a probability that the gene is either an oncogene or tumour suppressor gene. CGprio was considered to be in agreement with the aCGH data if it predicted a tumour suppressor gene in a region of copy number loss, or if it predicted an oncogene in a region of copy number gain.

4.3 Results

4.3.1 Quality control

After filtering to remove features with normalised \log_2 intensity values missing in more than 50% of the samples, the number of features in the dataset containing all the tumour samples was reduced by 3,381 leaving 28,273 or 89.31% of features from the 31,654 BAC clones on the Mermaid microarray with locations mapped to the hg18 assembly. In the good survival sample set, the number of features removed by this filtering step was 3,376 leaving 28,272 or 89.33% of features. In the poor survival sample set the number of features removed by this filtering step was 3,356 leaving 28,298 or 89.39% of features.

4.3.2 Recurrent region identification

The identification of recurrent regions of copy number variation was carried out using the KC-Smart algorithm. Run in parallel, the analysis of all nine cases took a little under 24 hours to complete, the vast majority of this computation time being taken up by the permutation analysis KS-smart performs which is required to calculate a significance value for the recurrent regions. The resulting totals of significant ($p < 0.01$) recurrent regions found in each of the three different data sets

Table 4.1: KC-Smart Regions Identified for each sample set and for each of the three kernel widths

Sample Set	2Mb Kernel		8Mb Kernel		24Mb Kernel		Present Across All Kernels	
	Losses	gains	Losses	gains	Losses	gains	Losses	gains
All (n=60)	46	44	27	21	40	41	44	43
Good (n=30)	25	13	12	12	34	37	23	10
Poor (n=30)	18	21	12	11	33	37	18	15

using the three different kernel widths can be seen in table 4.1. The high number of copy number gains and losses across all the samples is a clear reflection of the late stage tumour samples that were used, although there is very little difference in the number of copy number aberrations detected between the two different survival classes.

4.3.3 Significant recurrent copy number aberrations in stage III/IV serious ovarian tumours

The identification of recurrent regions of copy number change performed using the KC-smart algorithm on the entire dataset of 60 tumours provides us with an interesting view of the most common genomic regions lost and gained in stage III/IV serious ovarian tumours. Figure 4.2 shows the results of the KC-smart analysis using three differently sized kernels. The profiles show the KC score for each BAC on the microarray plotted according to genomic location, with different chromosomes coloured alternately and the additional red and green horizontal lines showing the significance cut-offs which are also calculated by the KC-smart method for the recurrent regions.

The use of three different kernel widths was an approach advocated by the

authors of the method, helping to detect potentially interesting recurrent regions that would otherwise be too small or large for detection using a single kernel approach. The effect of using different kernel widths is apparent in the profiles of Figure 4.2; the smallest kernel (2Mb) is capable of detecting smaller regions of recurrent aberration, producing a profile containing a large number of independent peaks representing multiple smaller recurrent regions. The KC score profile generated by the medium kernel (8MB) produced a smoother profile than the small kernel width, consistent with the kernel detecting large regions of recurrent aberration. This effect is accentuated further by the large kernel (24Mb) which is the smoothest of the three kernels with fewer peaks of recurrent aberration.

Overall the KC-smart profiles from the complete set of samples demonstrate a very high degree of genetic instability across the entire genome, with all chromosomes containing at least one significant recurrent region across all of the kernel widths with the exception of chromosome 19 4.2. There was no significant difference between the total number of aberrations which resulted in the gain of genetic material ($n=43$) and the total number of aberrations that resulted in a loss of genetic material ($n=44$).

4.3.4 Significant recurrent copy number aberrations in good prognosis tumours

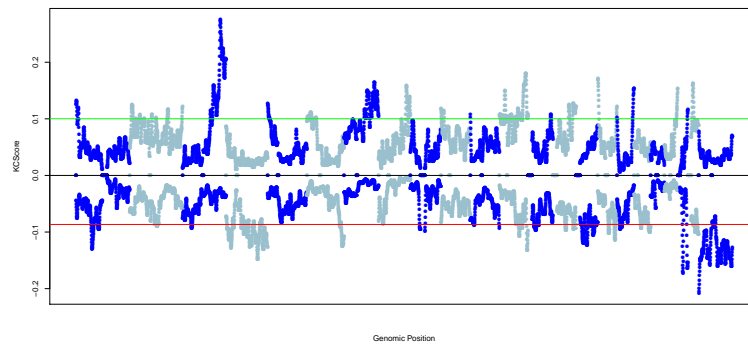
The KC-smart profiles produced from the set of 30 tumour samples which represented good patient survival shown in Figure 4.3 are similar to the profiles of all the samples combined in Figure 4.2, however the good survival KC-smart profiles have fewer regions reaching the statistical significance cut-off. There are highly significant regions of gain on chromosomes 3, 7, 8 and 16 and highly significant losses on chromosomes 1, 4, 21 and X.

4.3.5 Significant recurrent copy number aberrations in poor prognosis tumours

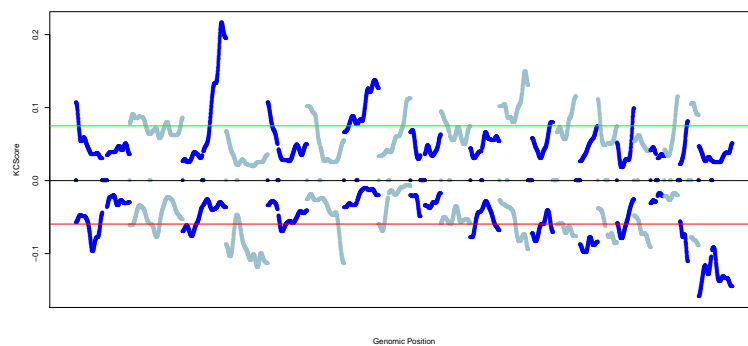
The KC-smart profiles produced from the set of 30 tumour samples which represented poor patient survival shown in Figure 4.4 are similar to both the good survival profiles and the combined set of all samples with a large number of recurrent copy number gains and losses. There is also similarity with the good survival profiles in the location of the recurrent aberrations, as the poor survival KC-smart profile shows significant recurrent gains on chromosomes 3 as in the good survival profile, there are also a number of less significant gains across a number of chromosomes not detected by the good survival results including 5, 12, 17 and 20. The KC-smart profile for the poor survival samples shows significant recurrent losses of genetic material on chromosomes 5, 17, and in similarity with the good survival results additional significant recurrent losses on chromosomes 21 and X. There was no difference observed in the total number of recurrent regions of CNV between the good and poor prognosis data sets as in both prognosis data sets 33 significant regions were identified.

4.3.6 Unique Breakpoint Identification

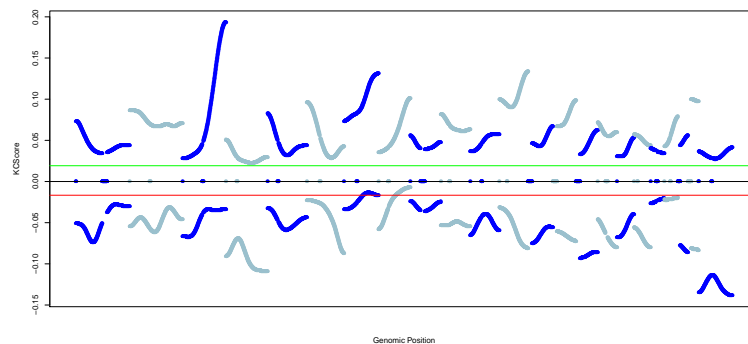
After analysis of the 60 experiments contained in the project we were able to identify 7630 unique minimal regions, this is a large reduction in the number of features describing the experiments in the project from an original number of 31654. We are therefore able to use less than a quarter of the features to fully study all the experiments, this reduction will allow more complex statistical analyses to be performed in much less time.



(a) KCSmart profile from all samples using a 2Mb kernel

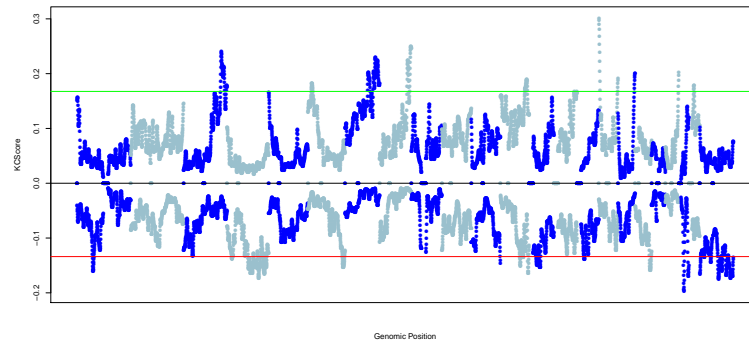


(b) KCSmart profile from all samples using a 8Mb kernel

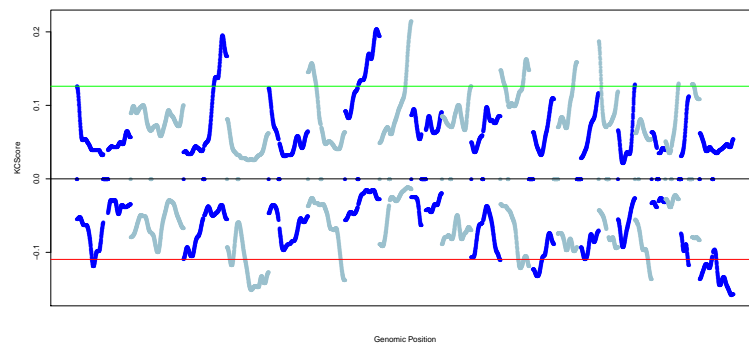


(c) KCSmart profile from all samples using a 24Mb kernel

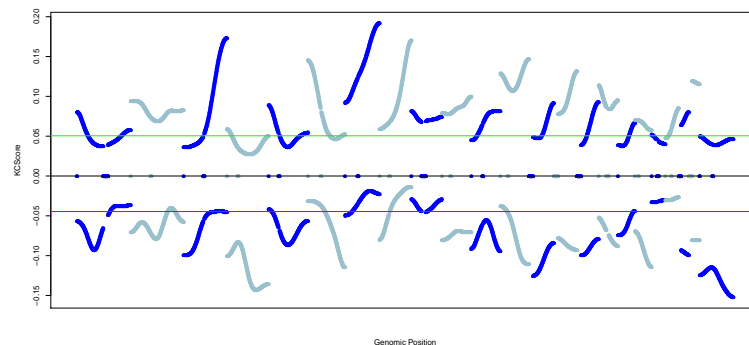
Figure 4.2: Output from KCSmart analysis of all 60 tumour samples at three different kernel widths. The KC score for every feature on the Mermaid aCGH microarray is plotted on the Y axis against the features genomic location on the X axis. The alternating light and blue colours represent chromosomes and the red and green lines are the significance thresholds calculated by KC-smart. Peaks below the red line are significantly recurrent copy number losses and peaks above the green line are significantly recurrent copy number gains.



(a) KCSmart profile from good survival samples using a 2Mb kernel

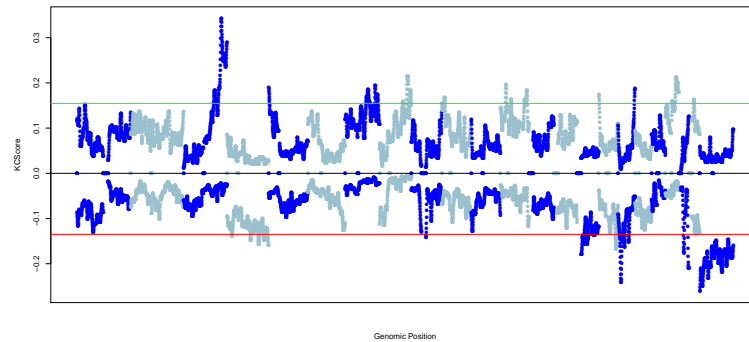


(b) KCSmart profile from good survival samples using an 8Mb kernel

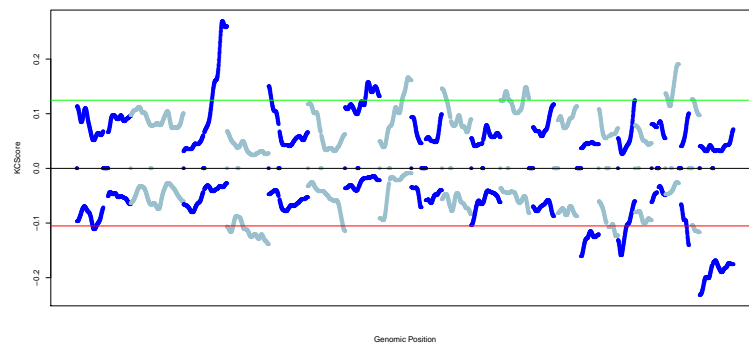


(c) KCSmart profile from good survival samples using a 24Mb kernel

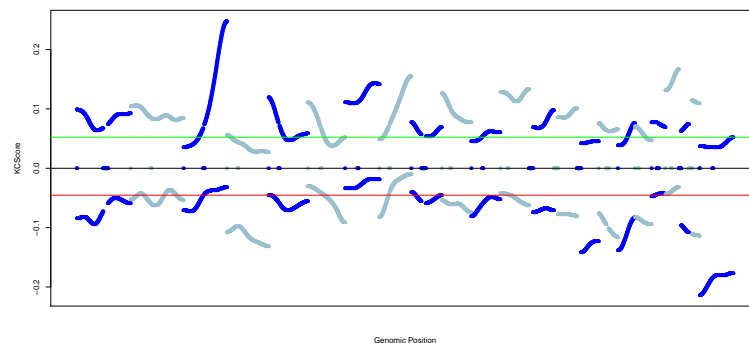
Figure 4.3: Output from KCSmart analysis of 30 good survival tumour samples at three different kernel widths. The KC score for every feature on the Mermaid aCGH microarray is plotted on the Y axis against the features genomic location on the X axis. The alternating light and blue colours represent chromosomes and the red and green lines are the significance thresholds calculated by KC-smart. Peaks below the red line are significantly recurrent copy number losses and peaks above the green line are significantly recurrent copy number gains.



(a) KCSmart profile from poor survival samples using a 2Mb kernel



(b) KCSmart profile from poor survival samples using an 8Mb kernel



(c) KCSmart profile from poor survival samples using a 24Mb kernel

Figure 4.4: Output from KCSmart analysis of 30 poor survival tumour samples at three different kernel widths. The KC score for every feature on the Mermaid aCGH microarray is plotted on the Y axis against the features genomic location on the X axis. The alternating light and blue colours represent chromosomes and the red and green lines are the significance thresholds calculated by KC-smart. Peaks below the red line are significantly recurrent copy number losses and peaks above the green line are significantly recurrent copy number gains.

4.3.7 Recurrent region feature selection

We applied our algorithm described in the recurrent region feature selection section of the methods of this chapter for identifying the unique significantly recurrent regions between the two survival classes using a variety of different cut off values for the percentage difference in average KC score between classes in order to select the best cut-off. The aim of this stage was to select the most discriminative regions and at the same time reduce the number of features taken forward to the next stage of analysis. The reduction in the number of features was required in order to make the the next analysis step, which is computationally intensive, feasible using the computing resources that were available to us.

A cut off level of 20% difference in the average KC score between classes reduces the number of features by over half; however the total number of possible combinations for 27 (134 million) was too high for analysis to be completed in a reasonable amount of time. At a difference of 20% there is a possibility of still including regions which are recurrent in both classes. Increasing the cut-off value to a minimum difference between classes of 25% reduced the number of unique regions further to 23 recurrent features. However, 23 features still generates more than 8 million possible combinations, which using our current hardware to assess the accuracy of each combination would take over 8 days. With our vision of the aCGH analysis pipeline being a system capable of quickly and easily re-analysing projects, an analysis step that would take over a week to complete would be unacceptable.

The difference cut-off in average KC score between classes that produced the best results in terms of reducing the number of features to a computationally manageable amount while keeping a good set of discriminative regions for further analysis was 33% which provided 16 regions, generating a manageable 65,535 combinations for

Table 4.2: KC-Smart features selected for classification

Region	Chromosome	Region Start (Mb)	Region End (Mb)	Region Size (Mb)	Aberration
11		133	134	1	Loss
12		89	93.5	4.5	Loss
12		103.5	105	1.5	Loss
13		32	40	8	Loss
13		47	53	6	Loss
16		0.5	6	5.5	Gain
18		66.5	74.5	8	Loss
10		0.5	1	0.5	Gain
10		11	15	4	Gain
15		19	34	15	Loss
15		68.5	69.5	1	Loss
17		7.5	22.5	15	Loss
17		28	32.5	4.5	Loss
20		31	34.6	3.6	Gain
20		45	62	17	Gain
23		1	58.5	57.5	Loss

analysis.

The analysis of all 65,535 combinations in fact took just under 24 hours to complete, splitting the analysis into 16 processes running in parallel (2 processes for each core of a dual quad core processor). As detailed in the materials and method section we applied this approach using a number of different input data sources, we were able to achieve the highest accuracy scores using the unique breakpoint regions and CGHcall data.

The highest leave one out cross validation accuracy across all 65,535 combinations was 83.33%, achieved by 1 single combination of 7 features. The specificity or true negative rate for this combination of features was 66.66% with a sensitivity or true positive rate of 90% and a Matthews Correlation Coefficient (MCC) of 0.57. The SVM classifier which achieved this performance, was created using CGHcall copy

Table 4.3: Best performing combination of KC-Smart features

Region	Chromosome	Region Start (Mb)	Region End (Mb)	Region Size (Mb)	Aberration
11		133	134	1	Loss
13		32	40	8	Loss
13		47	53	6	Loss
16		0.5	6	5.5	Gain
18		66.5	74.5	8	Loss
15		19	34	15	Loss
15		68.5	69.5	1	Loss

number calls of the unique segments as the variables in the recurrent regions.

The size and location of each of the 7 features is given in Table 4.3. The size of the 7 regions varies from two regions of 1Mb up to the largest region of 15Mb; the size of the regions is not smaller than a single mega base, as this is limited by the size of the smallest kernel (2Mb) used in the KC-smart recurrent region analysis step. The 7 regions occur on just 5 chromosomes, with two regions on chromosomes 13 and 15 and single regions on chromosomes 11, 16 and 18. Table 4.3 also displays the aberration type, from the combination of 7 regions only the single region on chromosome 16 was a recurrent gain in copy number, with the remaining 6 regions all recurrent copy number losses.

4.3.8 SVM Classifier Validation

The best performing classifier was validated using an independent set of 14 cases not included in the training set. The SVM was trained using data from all 60 cases from the training set, and then used to predict the survival status of the 14 test cases. The SVM correctly classified 12 of 14 cases, providing a classification accuracy of 85.7%.

Table 4.4: The significant recurrent regions ranked by number of contributions to accurate ($\geq 75\%$) SVM classifiers created using unique segments and CGHcall calls

Region	Chromosome	Region Start (Mb)	Region End (Mb)	Percentage of accurate combinations (n=234) containing region
16		0.5	6	89
13		47	53	70
15		68.5	69.5	65
15		19	34	63
18		66.5	74.5	61
20		45	62	57
20		31	34.5	54
10		0.5	1	52
11		133	134	50
12		89	93.5	45
10		11	15	43
17		28	32.5	41
13		32	40	39
12		10.35	10.5	29
17		7.5	22.5	17

4.3.9 Most informative regions

A measure of how informative each of the 16 recurrent regions used in the classification analysis was calculated by counting the number of times a region was used in an accurately classifying SVM (leave one out cross validation accuracy $\geq 75\%$). SVMs built using using the CGHcall call for every unique segment in the regions generated 234 different accurate combinations, the percentage of accurate combinations in which each of the 16 regions feature is listed in table 4.4. The table shows that the 5.5Mb region on chromosome 16 is the most important region in this analysis, as it contributes to 89% of accurate SVMs. In total 8 of the 16 regions contribute to more than 50%.

SVMs built using using the average \log_2 ratio for every microarray feature in

Table 4.5: The significant recurrent regions ranked by number of contributions to accurate ($\geq 75\%$) SVM classifiers created using BAC log2 ratios

Region	Chromosome	Region Start (Mb)	Region End (Mb)	Percentage of accurate combinations (n=92) containing region
16		0.5	6	91
15		19	34	88
20		45	62	83
10		11	15	77
11		133	134	70
13		47	53	68
18		66.5	74.5	66
12		89	93.5	52
12		10.35	10.5	49
10		0.5	1	48
13		32	40	43
20		31	34.5	34
15		68.5	69.5	29
17		7.5	22.5	23
17		28	32.5	21
X		1	58.5	1

the regions generated 92 different accurate combinations, the percentage of accurate combinations in which each of the 16 regions feature is listed in table 4.5. The table shows that the same region on chromosome 16 that was ranked first in the previous table, is also ranked first in this table because it is in 91% of the accurate SVMs.

Increasing the accuracy threshold for the SVMs to 80% for the analysis performed using the CGHcall call for every unique segment reduces the number of combinations, but it further confirms the previous results, as can be seen in table 4.6, as the region on chromosome 16 remains the most influential.

Table 4.6: The significant recurrent regions ranked by number of contributions to accurate ($\geq 80\%$) SVM classifiers created using unique segments and CGHcall calls

Region	Chromosome	Region Start (Mb)	Region End (Mb)	Percentage of accurate combinations (n=14) containing region
16		0.5	6	100
18		66.5	74.5	86
15		19	34	79
13		47	53	71
11		133	134	57
15		68.5	69.5	57
13		32	40	50
17		28	32.5	43
20		31	34.5	43
12		89	93.5	36
10		0.5	1	36
20		45	62	36
10		11	15	29
12		10.35	10.5	14

4.3.10 Gene Analysis

The most influential regions on SVM classification accuracy that were present in $>80\%$ of accurate SVMs were selected for further analysis in order to illustrate the utility of the previously described approach to identify interesting putative targets. The CGPrio resource described in the methods section was applied to the 4 regions in an attempt to reduce the number of possible targets genes in each region. Table 4.7 contains a summary of the CGprio results for each of the most informative KC-smart regions.

The 5.5Mb region on chromosome 16 contained 223 genes, this number was reduced to just 6 genes using a CGprio oncogene probability threshold of 0.9. The set of 6 genes contained 3 genes previously associated with cancer(*UBE2I*, *CREBBP* and *TFAP4*). The influential region on chromosome 15 contain the most genes at

249, however this was reduced to just 2 genes with a CGprio tumour suppressor probability greater than or equal to 0.9. The 2 genes were: *UBE3A* which has been associated cervical cancer along with a number of other diseases and *KLF13* which has been associated with erythroleukemia. The region on chromosome 18 contained the lowest number of genes at 28, with none of this smaller set of genes achieving a CGprio tumour suppressor probability greater than or equal to the threshold 0.9. The 17Mb region on chromosome 20 contained 211 genes, this was reduced to a more manageable 14 genes using a CGprio oncogene probability threshold of greater than or equal to 0.9. The 14 genes include 4 genes that have previously been associated with cancer (*ZMYND8*, *TNFRSF6B*, *GNAS* and *TAF4*) and 2 specifically with ovarian cancer (*NCOA3*, *PTK6*).

4.3.11 Random forests

Random forest classifiers were constructed as described in the methods section using the 16 regions identified in the recurrent region feature selection process on the data from the 60 samples that passed QC. The accuracy results of the random forests were disappointing, out of 1000 forests generated none produced an accuracy >50%. Based on these results and the performance of the SVM classifiers we decided to use SVMs in the large feature combination analysis to identify the most important regions for classification success. Our conclusion is supported by work comparing of random forests and support vector machines on microarray expression data in diagnostic and prognostic classification, which found that SVMs offer classification performance advantages compared to random forests (Statnikov *et al.*, 2008).

Table 4.7: Genes contained in the most influential classification regions

Chromosome	Region start (Mb)	Region end (Mb)	Total genes	Gene name	CGprio probability
16	0.5	6	223	<i>UBE2I</i>	0.9974
				<i>TRAF7</i>	0.9799
				<i>CASKIN1</i>	1.0
				<i>SRRM2</i>	0.9714
				<i>CREBBP</i>	1.0
				<i>TFAP4</i>	0.9966
18	66.5	74.5	28		
15	19	34	249	<i>UBE3A</i>	0.9013
				<i>KLF13</i>	0.9748
20	45	62	17	<i>ZMYND8</i>	0.9993
				<i>NCOA3</i>	1
				<i>MOCS3</i>	0.983
				<i>CEBPB</i>	0.999
				<i>PTPN1</i>	0.983
				<i>CBLN4</i>	991
				<i>TNFRSF6B</i>	0.999
				<i>SRMS</i>	0.973
				<i>PTK6</i>	0.994
				<i>RAB22A</i>	0.992
				<i>VAPB</i>	0.974
				<i>NPEPL1</i>	0.963
				<i>GNAS</i>	0.999
				<i>TAF4</i>	0.999

4.4 Conclusions

The work presented in this chapter has resulted in the identification of four very well supported genomic regions that appear to influence classification of patient survival through changes in copy number. These regions contain a number of interesting putative gene targets, a number are predicted to be associated with cancer and a number of those genes have actually been associated with cancer, adding weight to the case for these regions being important in ovarian cancer progression.

This work also demonstrated that the somatic genetic profile of an ovarian tumour measured by aCGH is a strong determinant of prognosis. Therefore confirming a genuine biological link between the copy number profile of a tumour and the survival period of the patient.

The approach applied here, using a recurrent region identification method to identify the best features to take forward for machine learning approaches; to then use the contribution made by each of the identified features in the machine learning process to pull out the most important regions for the biological phenomenon under study has worked extremely well. This has proved to be successful strategy for identifying putative targets for further investigation in the search for better understanding and therapeutic solutions for ovarian cancer.

Additionally, it can be concluded from the success of this project that the bioinformatics tools developed earlier in the project and described in the previous chapters successfully supported the informatics requirements of this project.

CHAPTER 5

Discussion

In this chapter I will discuss the results of the work presented in chapters 2, 3 and 4 of this thesis. I will also discuss where this work fits into the two research areas of bioinformatics and ovarian cancer research which it covers, and what contribution the thesis makes to work in both these fields of scientific research.

The aims of this project were to develop bioinformatics tools to support the management and analysis of high throughput data arising from microarray CGH experiments. These tools would then be used to investigate the genetic basis of patient survival in a study of ovarian tumour samples. Throughout the project, modern software development techniques were implemented at all stages of development, which in addition to the expected benefits of applying such practices would also help to demonstrate the suitability of such approaches to the field of bioinformatics.

At the end of this project all of the initial aims were successfully achieved; The first aim of creating bioinformatics tools for management and analysis tools for high throughput microarray CGH project was met primarily through the tools developed

and described in chapters 2 and 3.

The extension in functionality of the ArrayPipeline LIMS, which was the initial focus of work for the project, has proven to be an ideal solution for the management of high throughput microarray CGH data. Evidence for the success of the LIMS comes from the fact that the software is still currently in use in the laboratory and now contains data from more than 500 experiments. The success of these extensions to the LIMS functionality can be attributed to the software development techniques used. First, agile approaches towards responding to customer requirements were fundamental in ensuring the correct functionality was implemented as and when required. Since the entire aCGH project was still in development, this meant new features were requested and new builds released on a regular basis. Due to the schedule of regular incremental releases, the code base had to be flexible, so the use of version control was essential, documentation had to be of a high standard and constantly up to date. This process resulted in a LIMS which allowed users easily to record and track every step in the complex process of performing and analysing a microarray CGH experiment, as well as allowing users further to manage such experiments in the context of larger projects. In extending the ArrayPipeLine LIMS we also developed an easy to use object oriented programming toolkit that made creating new features and user interfaces very quick and simple.

The ArrayPipeLine LIMS was a successful solution also because all the data recorded in the ArrayPipeLine LIMS are stored in ways which ensure a high level of accuracy in the data. The methods used to verify data accuracy centred around the numerous quality control checks which were built into the LIMS, such as source code software tests and the use of a carefully structured relational database. The extensive use of relational database functionality, such as transaction safe tables, strict type checking and foreign key constraints are important for ensuring data integrity. For

instance, the use of transactions during the raw data processing process means that if any operation in this pipeline is not successfully completed, the database is able to ‘roll back’ to its previous state; and foreign key constraints are used extensively to enforce important relationships between data in the ArrayPipeLine database.

The third aim and central theme of this project was to illustrate how the use of formalised software development practices including agile techniques and other development best practices are suitable for bioinformatics projects, and that these approaches produce more successful results. I have demonstrated across the three results chapters the way in which modern software development techniques including agile were applied successfully in each task of the project. The success of each sub-project serves as clear proof that agile practices are highly suitable for bioinformatics applications, and that the software they produce is more fault tolerant, easier to maintain and overall more successful than software created without the use of formal development practices.

The suite of perl modules that allowed the simple and rapid generation of analysis pipelines which are described in chapter 3 satisfied the second part of the initial aim of the project, to develop tools for the analysis of high throughput data from microarray CGH data experiments. The analysis pipeline modules successfully harnessed two very powerful programming languages, Perl and R, together with analysis methods from the Bioconductor project, to create a framework for creating bespoke analysis pipelines. The analysis pipeline modules interact with the ArrayPipeLine LIMS for seamless data extraction and storage. The modules make it easy for users to track exactly what analysis a dataset has undergone, including all the parameters used, as every analysis step is recorded in the ArrayPipeLine LIMS database. Together, these features make the system extremely flexible and greatly simplify the re-analysis of entire data sets, while ensuring reproducibility and data

integrity.

The scalability of the management and analysis tools developed was identified as an important additional consideration, owing to the fact that the laboratory was aiming to profile a large number of samples using microarray CGH technology. Both the ArrayPipeLine LIMS and the analysis pipeline modules meet the required scalability demands through the use of development solutions whose scalability is well proven. The ArrayPipeLine LIMS uses object oriented Perl for the application logic, the Apache webserver to serve the web based user interface and a MySQL relational database for the backend storage. Perl, MySQL and the Apache web server are all well proven development tools that have been shown to be highly scalable through their use in many much larger projects. The analysis pipeline modules are highly flexible, making the incorporation of new analysis methods for larger datasets straightforward, and finally the use of object oriented programming techniques means it is very easy to expand the features of the modules.

The second aim of the project was to apply the bioinformatics tools developed for management and analysis of microarray CGH projects to some real data to demonstrate the ability of the system to cope with the demands of large number of cCGH experiments. The system was successfully applied to aCGH experiments carried out on just under one hundred ovarian cancer samples as part of an investigation attempting to identify recurrent copy number alteration of consequence associated with patient survival. The bioinformatics framework allowed the analysis of this project to be completed quickly and easily, with the extensive quality control measures available ensuring the quality of the data being analysed, thus providing greater confidence in the findings made.

5.1 Extension of the ArrayPipeLine LIMS

The work to extend the ArrayPipeLine LIMS resulted in a very successful solution, to the user demands that were made to support the laboratories aCGH experiment workflow. One very important choice of large impact made very early in the project was the decision to proceed with the development of our own bespoke LIMS. This decision obviously resulted in a good deal of time at the beginning of the project being spent on LIMS development. A valid question on this decision could be why time was spent on the development of a solution that already existed in commercial or freely available LIMS solutions? In order to answer this question it is necessary to define what was required in a LIMS solution for the laboratory which are described in detail in section 1.3.5 of the introduction. Briefly the essential features in a LIMS required by our laboratory were: support for the manufacture of aCGH microarrays; an analysis solution that provided flexibility to respond to new methods and approaches quickly; an API to enable higher level programming tasks against the data for higher level analysis; and support for the aCGH technology.

We assessed the available LIMS solutions at the beginning of the project, but no single application was available that was able to meet all our requirements. The solution which came closest to meeting these requirements was the analysis platform BASE previously mentioned in the introduction (Saal *et al.*, 2002). The BASE platform was a successful application for gene expression microarray data, however at the time this thesis began, it did not support either aCGH specific analysis, or crucially the microarray manufacturing process.

The ArrayPipeLine LIMS contribution to the aCGH community includes its use by a commercial company (Cambridge BlueGnome) for tracking their own aCGH manufacturing processes, and further contribution, beyond the aCGH community,

was made possible with the release of the core modules of the LIMS API on CPAN (Jones, 2008). This ‘toolbox’ of Perl source code created as a result of the work carried out extending the ArrayPipeLine LIMS, allows developers with basic knowledge of the Perl programming language to quickly and easily create their own LIMS applications (Morris *et al.*, 2008a).

The work carried out to develop the ArrayPipeLine LIMS also serves as an excellent example of how the informatics of high throughput projects should be organised. As it is essential to have a fully featured LIMS solution to manage the informatics demands that high throughput science brings, the ArrayPipeLine LIMS provides us with traceability, reproducibility and a high level of confidence in the results generated in the laboratory. With this in mind, the LIMS API has since been used rapidly to implement another LIMS in our Department, managing high-throughput SNP genotyping using the Fluidigm system (personal communication).

5.2 Analysis pipeline

The analysis pipeline modules are a flexible, robust and highly traceable system for pre-processing aCGH data. The pipeline modules do however have some limitations. In this section I will discuss these limitations and also discuss some of the design decisions that were taken in the process of their development.

The analysis pipeline suite of modules does not include modules for all the available aCGH analysis methods in the Bioconductor project, but instead only specific methods were selected and implemented, based on their suitability with the data being generated in the laboratory and the published assessments of the best performing methods for certain analysis steps.

The authors of the MANOR package reported that the most common source

of bias in the aCGH experiments they witnessed were spacial artefacts, and the MANOR algorithm was designed to identify and either remove or normalise such artefacts. The MANOR package was also the only available aCGH specific normalisation package available at the time the analysis pipeline modules were developed, with the vast majority of normalisation methods designed specifically for expression arrays.

We felt it was necessary to apply a normalisation method specific to aCGH, because aCGH data has important differences compared with gene expression microarray data. The most obvious difference between the two technologies is the dynamic range of the data, with aCGH aiming to capture single copy number increases or decreases in segments of DNA whereas gene expression experiments aim to capture enrichment in gene expression which can be thousands of times higher than the reference. However, it was clear that signal bias was also a common issue with our aCGH experiments, so we also implemented an analysis pipeline module for signal correction using the loess package in Bioconductor.

The choice of which segmentation method to implement in the analysis pipeline suite required assessment of the available methods. This analytical challenge has been the most researched in aCGH informatics: there are a number of different methods available to perform segmentation of aCGH data. We considered the results of two published reviews which assessed the performance of a number of methods. Both reviews identified DNACopy as a method which performed very well, having the best operational characteristics in terms of its sensitivity and FDR for breakpoint detection (Willenbrock and Fridlyand, 2005). The CGHcall method for assigning segments a discrete copy number status was chosen for inclusion in the analysis pipeline because it was the only method at the time capable of performing such an analysis that also provided probability scores for each status and was also compatible

with the output format from the DNACopy segmentation method.

The analysis pipeline modules implement methods (MANOR, loess, DNACopy, CGHcall) that process the raw intensity signals from an aCGH experiment into discretized copy number values. It has however been suggested in some papers that by processing the raw experimental signal in such a way, interesting biological signals are lost and the best solution is in fact to use the raw intensity data for identification of recurrent regions from multiple samples. Discretization approaches, because they employ a cut-off to assign a copy number status based on, for example, probability values, will always result in situations where a genuinely aberrant segment of DNA is called as normal because it is just under the threshold; or a normal segment of DNA is called as aberrant due to being just over the given threshold.

While the above might be true of high quality data, the data from FFPE is generally noisy and of highly variable quality. As a result, we considered pre-processing necessary in order to standardise the quality between experiments. However our downstream analysis was not concerned with detecting the smallest changes, or as many changes as possible, but rather at establishing common profiles that might be specific to histopathological and clinical features of the disease. For this reason, the analysis method implemented in the KCsmart package which uses raw data was chosen for downstream analysis. Processing of the data still remains vitally important for visualisation.

Work further to extend the functionality and usability of the suite of analysis pipeline modules would include implementation of new analysis methods published since the end of this thesis. New methods could be added to the analysis pipeline suite of modules very easily and entire data sets re-analysed with little effort, if required.

To make the analysis pipeline suite of modules available to others would require

some further development to make the modules less dependant on the ArrayPipeLine LIMS database. A more generic system could use any simple analysis script, not necessarily just R scripts or shell scripts, to launch other software solutions. A generic set of modules could then be added to CPAN for wider use beyond just aCGH. This work could be carried out in combination with work on the underlying code base to utilise use of new technologies which make object oriented development even easier, such as using the Perl object system Moose (Infinity Interactive, 2011) for creating and maintaing Perl objects.

An analysis pipeline created using the analysis pipeline modules currently runs as a single process and does not make use of modern multicore processors or compute farms. However it would be trivial to implement a system to dispatch individual tasks in a pipeline as jobs to a compute farm, or on a cloud based service.

5.3 MALOVA

The analysis of the 94 stage III/IV ovarian tumour samples described in chapter 4 of this thesis yielded a number of interesting results. These profiles confirmed our expectations that these advanced stage tumours would contain many chromosomal aberrations, with nearly all chromosomes containing at least one significant recurrent region of gain or loss.

The analysis of these data also identified 16 significantly recurrent regions which showed different representation between good and poor survival patients. Four of these recurrent regions were independently identified in a study which looked for associations between copy number aberration and chemotherapy resistance in a similar data set to mine (Etemadmoghadam *et al.*, 2009). Etemadmoghadam *et al.* identified recurrent copy number aberration from 118 late stage serous

Table 5.1: Regions of overlap with Etemadmoghadam *et al*

Aberration	Chromosome	Region(Mb)	Region Size (Mb)	Etemadmoghadam Region (Mb)	Etemadmoghadam Region Size (Mb)
Gain	20	45-62	17	55.64-56.45	0.8
Loss	12	89-93.5	4.5	86.72-86.75	0.03
Loss	13	47-53	6	47-52.88	5.9
Loss	18	66.5-74.5	8	61.9-64.08	2.2

ovarian tumours using high-resolution oligonucleotide microarrays. They found that amplification of 19q12, containing cyclin E (*CCNE1*), and 20q11.22-q13.12, mapping immediately adjacent to the steroid receptor coactivator *NCOA3*, was significantly associated with poor response to primary treatment (Etemadmoghadam *et al.*, 2009). While my analysis did not identify the 19q12 region, it did confirm the region of gain on 20q, identifying a slightly larger and completely overlapping region to the one described by Etemadmoghadam *et al.* The three other regions of similarity between our two studies are regions of loss on chromosome 12, chromosome 13 and chromosome 18 (see Table 5.1).

I was then able to construct an SVM classifier that used aCGH data from the 16 recurrent regions identified, to predict the survival status of a patient with an accuracy of 83.3%. This classifier was then validated with an accuracy of 85.7%, using independent data to confirm there were no problems associated with overfitting. However, the approach used to identify the most accurate SVM classifier involved testing every possible combination of the 16 recurrent regions, and hence introduces the significant issue of multiple testing; by testing over 60,000 SVM classifiers, we would expect a number to provide accurate classifications by chance. While this problem is alleviated to some extent by validation of the classifier with independent data, I decided also to consider the contributions each of the regions

made to the most accurate classifiers. Ranking the recurrent regions used in the SVM analysis by how often they contribute to accurate classification helps to overcome the issues of multiple testing and over-fitting, since it does not test individual results.

The ability to classify patient survival based on a DNA copy number profile has some clinical utility. However, a more important outcome of this analysis is that the regions identified as being associated with survival would clearly represent potential therapeutic targets. This is supported by the identification of a number of genes that have already been associated with cancer from the four most influential regions for SVM classifier accuracy.

A clear limitation of this work is the small sample size of just under one hundred cases. This will have directly impacted on the sensitivity of detecting regions of gain/loss and better refining their boundaries, as well as the ability accurately to train/test a SVM. Following the completion of this thesis, work has continued in the laboratory and much of the analysis work I have presented has been repeated with larger sample sets. Interestingly, this has resulted in the identification of the *CCNE1* locus on chromosome 19 as one of the most influential regions for survival classification, confirming the earlier findings of Etemadmoghadam *et al* (manuscript in preparation). In addition, the data published by Etemadmoghadam *et al* will be used to validate the results of the most recent work, providing greater confidence in the results generated.

5.4 Bioinformatics software development practices

The issue of software development practices in bioinformatics remains very important and will become more important with second and third generation sequencing technologies. While this issue has been neglected in the past, it is now

receiving attention from many researchers, there are also a number of exemplary bioinformatics projects demonstrating the power of using software development best practices for bioinformatics.

However practices only deliver their full value when they are shared among all practitioners, and unfortunately for the field of bioinformatics this is not the case. Even the most basic techniques are not routinely used even in a leading bioinformatics research institute. For instance, in a small survey of forty computationally based researchers in Human Genetics at the Wellcome Trust Sanger Institute, less than a quarter use a version control system. The personal experiences of several colleagues suggests that other practices such as software testing and inline documentation are similarly poorly applied across the field. The lack of clear standards in bioinformatics software development will inevitably result in poorly functioning software and results that are difficult to reproduce and potentially incorrect.

One possible mechanism for wider adoption of basic software best practices would be through the use of source code reviews in the publication process. A recent letter in the journal *Nature* suggests that all the programming code associated with a published papers should also be published (Barnes, 2010). This approach would both encourage scientists to write better quality code and allow a greater level of peer review of source code, which is currently lacking in the scientific publication process.

If researchers working in the field of bioinformatics were to adopt more rigorous software development practices, then far more bioinformatics projects would generate high quality software that meets user's requirements and produce accurate, reproducible results. In addition, early career training in such practices would yield huge gains in productivity from developers, benefitting both individual projects and

the entire bioinformatics community.

In summary, the most basic best practices that every bioinformatics project should attain are as follows;

- Testing to ensure accurate results
- Thorough documentation for reproducibility
- Version control of source code
- Re-use source code where suitable solutions are available, or develop software in a manner to encourage the generation of reusable code such as object oriented programming
- Customer interaction to ensure the delivery of suitable software, where appropriate

When using version control systems to manage the source code for a project it is important to make use of more advanced features such as version tagging or labelling. Labelling is a feature of most VCSs, it allows developers to associate a symbolic tag to a particular set of revisions, such as a significant release of software. This is a very useful technique because the revision numbers used by VCSs do not record information on the software release version. I encountered a great deal of difficulty during the writing of this thesis in attempting to use early versions of the ArrayPipeLine LIMS. This task was made challenging because the revision numbers of individual files do not relate to significant steps in the projects development. Therefore to be able to run an older revision of the software needed knowledge of all the required revision numbers of all the dependant files, as the number of files that each part of the ArrayPipeLine LIMS depends on is very large this was an extremely difficult task. However, running an older revision of software

would be relatively trivial had revision tags been used in the development process, thus allowing us to recreate the software at each release step.

Many of the agile approaches such as the planning game, pair programming and an on-site customer from extreme programming and the 30 day sprint and daily meetings from scrum are unlikely to be of real benefit in an academic software development setting. Agile also assumes that a development task can not be understood and fully designed and planned for up front which in many cases is wrong. Using an agile approach in such a situation could delay the delivery of a solution due to the necessity to regularly discuss requirements with customers. Agile approaches also do not help in the very common situation where a major decision needs to be made at the beginning of a project, such as whether to use an existing solution or build a new one. However overall it is clear that a number of the best practices promoted by the agile movement are perfectly suited to the field of bioinformatics.

5.5 Future prospects

A clear priority for the future prospects of this work is the continued identification of new ovarian cancer genes. New targets for potential therapeutic targets and prognostic markers are still required. The process of identifying these new genes however is quickly moving to away from aCGH and towards sequencing of entire cancer genomes using next generation sequencing technologies (The International Cancer Genome Consortium, 2010). Identifying copy number aberrations from sequence data will require new processing techniques as this technology produces counts of reads as an output and not intensity values. Dimension reduction may also be necessary to relieve the computational burden for downstream analysis (van de Wiel *et al.*, 2011).

One clear advantage and exciting opportunity of moving away from aCGH to sequencing based investigations, will be the ability to detect the genomic alterations that are not detected by aCGH. Alterations such as mutations, SNPs, unbalanced translocations and inversions have been neglected using aCGH, but will now be fully studied in the hope of detecting novel causative variants for cancer.

An interesting future prospect of bioinformatics development is also closely tied to the rise of low cost genomic sequencing. Second and third generation sequencing technologies will provide some of the largest bioinformatics challenges in the near future. This is because that the rate of sequence production, and therefore, the requirements for data storage and processing power is outstripping Moore's law for microprocessors, which states that computational power will double every 18 months. This means that in small and medium laboratories it is becoming very hard to purchase and maintain the levels of computational power necessary to work with large volumes of sequencing data.

One possible answer to this problem, that is gathering strength, is the use of cloud computing. Cloud computing services like those provided by Amazon's Elastic Compute Cloud (EC2) service provide scalable computing services on remote computing clusters allowing users to pay only for the resources they need. This makes it possible for a small laboratory to gain access to a large compute cluster, for a short period of time, without the costly expense of creating and maintaining their own physical cluster.

Cloud technologies will also make packaging and distributing software and pipelines for use by other groups far easier as in the past this has been challenging due to software dependencies and site-specific configuration options. In a cloud computing environment these pipelines can be packaged into virtual machine images and stored in a way that lets anyone copy them, run them and customise them

for their own needs, thus avoiding the software installation and configuration complexities. A virtual machine is a piece of software running on the host computer that emulates the properties of a computer (Stein, 2010).

5.6 Concluding remarks

This thesis makes contributions to both the fields of bioinformatics and ovarian cancer research. To the field of bioinformatics, this thesis adds a number of high quality tools for the management and analysis of high throughput data. This statement is upheld through the availability of a number of Perl modules on CPAN and the fact that the ArrayPipeLine LIMS and analysis pipeline code is still being successfully used in the laboratory today. Also for the field of bioinformatics, I was able to argue and demonstrate the importance and value of using formalised software development practices in bioinformatics project development. Finally in the field of ovarian cancer research, using the bioinformatics tools described in this thesis, I was able to show that the aCGH profile of an ovarian tumour can be used to identify important genomic regions which contribute to patient survival. This work will hopefully lead to further identification and characterisation of novel genes that affect the development of ovarian cancer and hopefully yield possible clinical applications against a disease that has not seen any significant increase in survival rates over the past 50 years.

In considering the different approaches that could have been taken to complete this project, it is very easy to identify a number of different applications for Perl development that are now becoming widely adopted in production level projects. Web development frameworks like Catalyst (Catalyst Foundation, 2012), Mojo (Riedel, 2012), CGI::Application (Erlbaum, 2012) and Jifty (Vincent *et al.*, 2012)

which automate a great deal of the repetitive tasks of website development would have made development of the LIMS easier. The DBIx::Class (Trout, 2012) object relational mapper for Perl makes working with data in databases much easier, making it an ideal tool for the LIMS API. The MOOSE object system (Infinity Interactive, 2011) for Perl which is currently becoming popular would have made the development of the array pipeline modules an even easier task.

The central theme to this thesis, which was the importance of formalised software development practices for bioinformatics application development, was independent of the technology chosen to develop the resulting tools. Therefore the choice of programming languages and other software solutions while important was not critical, as it was the development practices (testing, customer interaction) used that had the largest impact on the quality of software produced.

Bibliography

- Amaratunga, D., Cabrera, J., and Lee, Y.-S. (2008). Enriched random forests. *Bioinformatics*, **24**(18), 2010–2014.
- Ambler, S. W. (2010). Introduction to data normalization: A database ”best” practice. <http://www.agiledata.org/essays/dataNormalization.html>.
- Bain, S. L. (2008). *Emergent Design: The Evolutionary Nature of Professional Software Development*. Addison-Wesley Professional, 1 edition.
- Barnes, N. (2010). Publish your computer code: it is good enough. *Nature*, **467**(7317), 753.
- Bast, R. C., Hennessey, B., and Mills, G. B. (2009). The biology of ovarian cancer: new opportunities for translation. *Nature reviews. Cancer*, **9**(6), 415–428.
- Baxter, S. M., Day, S. W., Fetrow, J. S., and Reisinger, S. J. (2006). Scientific software development is not an oxymoron. *PLoS Comput Biol*, **2**(9), e87+.
- Beck, K. (1999). Embracing change with extreme programming. *Computer*, **32**(10), 70–77.
- Benedet, J. L., Bender, H., Jones, H., Ngan, H. Y., and Pecorelli, S. (2000). FIGO staging classifications and clinical practice guidelines in the management of

gynecologic cancers. FIGO Committee on Gynecologic Oncology. *International journal of gynaecology and obstetrics: the official organ of the International Federation of Gynaecology and Obstetrics*, **70**(2), 209–262.

Bentley, D. R., Deloukas, P., Dunham, A., French, L., Gregory, S. G., Humphray, S. J., Mungall, A. J., Ross, M. T., Carter, N. P., Dunham, I., Scott, C. E., Ashcroft, K. J., Atkinson, A. L., Aubin, K., Beare, D. M., Bethel, G., Brady, N., Brook, J. C., Burford, D. C., Burrill, W. D., Burrows, C., Butler, A. P., Carder, C., Catanese, J. J., Clee, C. M., Clegg, S. M., Cobley, V., Coffey, A. J., Cole, C. G., Collins, J. E., Conquer, J. S., Cooper, R. A., Culley, K. M., Dawson, E., Dearden, F. L., Durbin, R. M., de Jong, P. J., Dhami, P. D., Earthrowl, M. E., Edwards, C. A., Evans, R. S., Gillson, C. J., Ghorri, J., Green, L., Gwilliam, R., Halls, K. S., Hammond, S., Harper, G. L., Heathcote, R. W., Holden, J. L., Holloway, E., Hopkins, B. L., Howard, P. J., Howell, G. R., Huckle, E. J., Hughes, J., Hunt, P. J., Hunt, S. E., Izmajlowicz, M., Jones, C. A., Joseph, S. S., Laird, G., Langford, C. F., Lehtvaslaiho, M. H., Leversha, M. A., McCann, O. T., McDonald, L. M., McDowall, J., Maslen, G. L., Mistry, D., Moschonas, N. K., Neocleous, V., Pearson, D. M., Phillips, K. J., Porter, K. M., Prathalingam, S. R., Ramsey, Y. H., Ranby, S. A., Rice, C. M., Rogers, J., Rogers, L. J., Sarafidou, T., Scott, D. J., Sharp, G. J., Shaw-Smith, C. J., Smink, L. J., Soderlund, C., Sotharan, E. C., Steingruber, H. E., Sulston, J. E., Taylor, A., Taylor, R. G., Thorpe, A. A., Tinsley, E., Warry, G. L., Whittaker, A., Whittaker, P., Williams, S. H., Wilmer, T. E., Wooster, R., and Wright, C. L. (2001). The physical maps for sequencing human chromosomes 1, 6, 9, 10, 13, 20 and X. *Nature*, **409**(6822), 942–943.

biojava.org (2012). Biojava. <http://biojava.org/>.

bioperl.org (2011). Bioperl. <http://www.bioperl.org/>.

biopython.org (2012). Biopython. <http://biopython.org/>.

Brazma, A., Parkinson, H., Sarkans, U., Shojatalab, M., Vilo, J., Abeygunawardena, N., Holloway, E., Kapushesky, M., Kemmeren, P., Lara, G. G. G., Oezcimen, A., Rocca-Serra, P., and Sansone, S.-A. A. (2003). Arrayexpress—a public repository

- for microarray gene expression data at the ebi. *Nucleic acids research*, **31**(1), 68–71.
- Breiman, L. (2001). Random forests. *Machine Learning*, **45**, 5–32. 10.1023/A:1010933404324.
- Breiman, L., Cutler, A., Liaw, A., and Wiener, M. (2012). randomforest: Breiman and Cutler’s random forests for classification and regression. <http://cran.r-project.org/web/packages/randomForest/index.html>.
- Bunce, T. (2012). Dbi.pm. <http://search.cpan.org/timb/DBI-1.622/DBI.pm>.
- Canonical (2012). Storm. <https://storm.canonical.com/>.
- Carter, N. P. (2007). Methods and strategies for analyzing copy number variation using DNA microarrays. *Nature Genetics*, **39**(7 Suppl), S16–S21.
- Carvalho, B., Ouwerkerk, E., Meijer, G. A., and Ylstra, B. (2004). High resolution microarray comparative genomic hybridisation analysis using spotted oligonucleotides. *Journal of clinical pathology*, **57**(6), 644–646.
- Catalyst Foundation (2012). Catalyst web framework: Perl mvc framework. <http://www.catalystframework.org/>.
- Chacon, S. (2012). Git. <http://git-scm.com/>.
- Chin, K., DeVries, S., Fridlyand, J., Spellman, P. T., Roydasgupta, R., Kuo, W.-L., Lapuk, A., Neve, R. M., Qian, Z., and Ryder, T. (2006). Genomic and transcriptional aberrations linked to breast cancer pathophysiologies. *Cancer Cell*, **10**(6), 529–541.
- Christersson, J. V., Nordborg, N., Svensson, M., and Hakkinen, J. (2009). Base - 2nd generation software for microarray data management and analysis. *BMC Bioinformatics*, **10**(1), 330+.
- Cockburn, A. (2004). *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison-Wesley Professional, 1 edition.

- Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, **13**(6), 377–387.
- Codd, E. F. (1971a). Further normalization of the data base relational model. Technical Report RJ909, IBM.
- Codd, E. F. (1971b). Normalized data base structure: a brief tutorial. In *Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '71, pages 1–17, New York, NY, USA. ACM.
- Conrad, D. F., Pinto, D., Redon, R., Feuk, L., Gokcumen, O., Zhang, Y., Aerts, J., Andrews, T. D., Barnes, C., Campbell, P., Fitzgerald, T., Hu, M., Ihm, C. H., Kristiansson, K., MacArthur, D. G., MacDonald, J. R., Onyiah, I., Pang, A. W., Robson, S., Stirrups, K., Valsesia, A., Walter, K., Wei, J., Tyler-Smith, C., Carter, N. P., Lee, C., Scherer, S. W., and Hurles, M. E. (2009). Origins and functional impact of copy number variation in the human genome. *Nature*, **464**(7289), 704–712.
- Consortium, B. A. C. R., Cheung, V. G., Nowak, N., Jang, W., Kirsch, I. R., Zhao, S., Chen, X. N., Furey, T. S., Kim, U. J., Kuo, W. L., Olivier, M., Conroy, J., Kasprzyk, A., Massa, H., Yonescu, R., Sait, S., Thoreen, C., Snijders, A., Lemyre, E., Bailey, J. A., Bruzel, A., Burrill, W. D., Clegg, S. M., Collins, S., Dhami, P., Friedman, C., Han, C. S., Herrick, S., Lee, J., Ligon, A. H., Lowry, S., Morley, M., Narasimhan, S., Osoegawa, K., Peng, Z., Plajzer-Frick, I., Quade, B. J., Scott, D., Sirotkin, K., Thorpe, A. A., Gray, J. W., Hudson, J., Pinkel, D., Ried, T., Rowen, L., Shen-Ong, G. L., Strausberg, R. L., Birney, E., Callen, D. F., Cheng, J. F., Cox, D. R., Doggett, N. A., Carter, N. P., Eichler, E. E., Haussler, D., Korenberg, J. R., Morton, C. C., Albertson, D., Schuler, G., de Jong, P. J., and Trask, B. J. (2001). Integration of cytogenetic landmarks into the draft sequence of the human genome. *Nature*, **409**(6822), 953–958.
- Consortium, D. (2012). DSDM consortium. <http://www.dsdm.org/>.
- Conway, D. (2000). *Object Oriented Perl: A Comprehensive Guide to Concepts and Programming Techniques*. Manning Publications.

- cplusplus.com (2012). C++ language. <http://www.cplusplus.com/>.
- cran.r project.org (2012). Comprehensive R Archive Network. <http://cran.r-project.org/>.
- cran.r project.org (2012). gtools: Various R programming tools. <http://cran.r-project.org/web/packages/gtools/index.html>.
- Cunningham, W. (2001). The Agile Manifesto. <http://agilemanifesto.org/>.
- Dafou, D., Ramus, S. J., Choi, K., Grun, B., Trott, D. A., Newbold, R. F., Jacobs, I. J., Jones, C., and Gayther, S. A. (2009). Chromosomes 6 and 18 induce neoplastic suppression in epithelial ovarian cancer cells. *Int. J. Cancer*, **124**(5), 1037–1044.
- Dasarathy, B. V., editor (1990). *Nearest Neighbor: Pattern Classification Techniques (Nn Norms : Nn Pattern Classification Techniques)*. Ieee Computer Society.
- Diskin, S. J., Eck, T., Greshock, J., Mosse, Y. P., Naylor, T., Stoeckert, C. J., Weber, B. L., Maris, J. M., and Grant, G. R. (2006). Stac: A method for testing the significance of DNA copy number aberrations across multiple array-CGH experiments. *Genome Research*, **16**(9), 1149–1158.
- DuBois, P. (2008). *MySQL (4th Edition)*. Addison-Wesley Professional, 4 edition.
- Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Wiley & Sons, first edition.
- Dudley, J. T. and Butte, A. J. (2009). A quick guide for developing effective bioinformatics programming skills. *PLoS Comput Biol*, **5**(12), e1000589+.
- Eilers, P. H. C. and de Menezes, R. X. (2005). Quantile smoothing of array CGH data. *Bioinformatics*, **21**(7), 1146–1153.
- ensembl.org (2012). Ensembl perl api documentation. <http://www.ensembl.org/info/docs/api/index.html>.

- Erlbaum, J. (2012). Cgi::application. <http://cgi-app.org/>.
- Etemadmoghadam, D., deFazio, A., Beroukhi, R., Mermel, C., George, J., Getz, G., Tothill, R., Okamoto, A., Raeder, M. B., Group, A. S., Harnett, P., Lade, S., Akslen, L. A., Tinker, A. V., Locandro, B., Alsop, K., Chiew, Y.-E., Traficante, N., Feraday, S., Johnson, D., Fox, S., Sellers, W., Urashima, M., Salvesen, H. B., Meyerson, M., and Bowtell, D. (2009). Integrated genome-wide dna copy number and expression analysis identifies distinct mechanisms of primary chemoresistance in ovarian carcinomas. *Clinical Cancer Research*, **15**(4), 1417–1427.
- Evgenia Dimitriadou, Kurth, D. M. and Andreas Weinges Hornik, F. L. (2011). e1071: Misc functions of the department of statistics (e1071), tu wien. <http://cran.r-project.org/web/packages/e1071/index.html>.
- Fagin, R. (1977). Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst.*, **2**(3), 262–278.
- Fagin, R. (1979). Normal forms and relational database operators. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, SIGMOD '79, pages 153–160, New York, NY, USA. ACM.
- Fridlyand, J. (2004). Hidden Markov models approach to the analysis of array CGH data. *Journal of Multivariate Analysis*, **90**(1), 132–153.
- Furmento, N., Mayer, A., McGough, S., Newhouse, S., Field, T., and Darlington, J. (2002). ICENI: Optimisation of component applications within a Grid environment. *Parallel Computing*, **28**(12), 1753–1772.
- Furney, S. J., Calvo, B., Larrañaga, P., Lozano, J. A., and Lopez-Bigas, N. (2008). Prioritization of candidate cancer genes—an aid to oncogenomic studies. *Nucleic acids research*, **36**(18), e115+.
- Geeknet, Inc. (2012). sourceforge. <https://sourceforge.net/>.
- Gentleman, R., Carey, V., Bates, D., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., Hornik, K., Hothorn, T., Huber, W., Iacus, S.,

- Irizarry, R., Leisch, F., Li, C., Maechler, M., Rossini, A., Sawitzki, G., Smith, C., Smyth, G., Tierney, L., Yang, J., and Zhang, J. (2004). Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, **5**(10), R80+.
- Giardine, B., Riemer, C., Hardison, R. C., Burhans, R., Elnitski, L., Shah, P., Zhang, Y., Blankenberg, D., Albert, I., Taylor, J., Miller, W., Kent, W. J., and Nekrutenko, A. (2005). Galaxy: a platform for interactive large-scale genome analysis. *Genome research*, **15**(10), 1451–1455.
- GitHub Inc (2012). Github. <https://github.com/>.
- Guttman, M., Mies, C., Dudycz-Sulicz, K., Diskin, S. J., Baldwin, D. A., Stoeckert, C. J., and Grant, G. R. (2007). Assessing the Significance of Conserved Genomic Aberrations Using High Resolution Genomic Microarrays. *PLoS Genet*, **3**(8), e143+.
- Haverty, P., Hon, L., Kaminker, J., Chant, J., and Zhang, Z. (2009). High-resolution analysis of copy number alterations and associated expression changes in ovarian tumors. *BMC Medical Genomics*, **2**(1), 21+.
- hibernate.org (2012). HIBERNATE. <http://www.hibernate.org/>.
- Highsmith, J. (2002). *Agile Software Development Ecosystems*. Addison-Wesley Professional.
- Highsmith, J. A. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House.
- Hinrichs, A. S., Karolchik, D., Baertsch, R., Barber, G. P., Bejerano, G., Clawson, H., Diekhans, M., Furey, T. S., Harte, R. A., Hsu, F., Hillman-Jackson, J., Kuhn, R. M., Pedersen, J. S., Pohl, A., Raney, B. J., Rosenbloom, K. R., Siepel, A., Smith, K. E., Sugnet, C. W., Sultan-Qurraie, A., Thomas, D. J., Trumbower, H., Weber, R. J., Weirauch, M., Zweig, A. S., Haussler, D., and Kent, W. J. (2006). The UCSC Genome Browser Database: update 2006. *Nucleic Acids Research*, **34**(suppl 1), D590–D598.

- Hodgson, G., Hager, J. H., Volik, S., Hariono, S., Wernick, M., Moore, D., Albertson, D. G., Pinkel, D., Collins, C., Hanahan, D., and Gray, J. W. (2001). Genome scanning with array CGH delineates regional alterations in mouse islet carcinomas. *Nature Genetics*, **29**(4), 459–464.
- Høgdall, E. V., Christensen, L., Kjaer, S. K., Blaakaer, J., Bock, J. E., Glud, E., Nørgaard-Pedersen, B., and Høgdall, C. K. (2003). Distribution of HER-2 overexpression in ovarian carcinoma tissue and its prognostic value in patients with ovarian carcinoma: from the Danish MALOVA Ovarian Cancer Study. *Cancer*, **98**(1), 66–73.
- Høgdall, E. V., Ryan, A., Kjaer, S. K., Blaakaer, J., Christensen, L., Bock, J. E., Glud, E., Jacobs, I. J., and Høgdall, C. K. (2004). Loss of heterozygosity on the X chromosome is an independent prognostic factor in ovarian carcinoma: from the Danish "MALOVA" Ovarian Carcinoma Study. *Cancer*, **100**(11), 2387–2395.
- Høgdall, E. V. S., Christensen, L., Kjaer, S. K., Blaakaer, J., Kjærbye-Thygesen, A., Gayther, S., Jacobs, I. J., and Høgdall, C. K. (2007). CA125 expression pattern, prognosis and correlation with serum CA125 in ovarian tumor patients. *Gynecologic Oncology*, **104**(3), 508–515.
- Hoon, S., Ratnapu, K. K., Chia, J.-m., Kumarasamy, B., Juguang, X., Clamp, M., Stabenau, A., Potter, S., Clarke, L., and Stupka, E. (2003). Biopipe: A Flexible Framework for Protocol-Based Bioinformatics Analysis. *Genome Research*, **13**(8), 1904–1915.
- Hsu, L., Self, S. G., Grove, D., Randolph, T., Wang, K., Delrow, J. J., Loo, L., and Porter, P. (2005). Denoising array-based comparative genomic hybridization data using wavelets. *Biostatistics*, **6**(2), 211–226.
- Huang, H., Nguyen, N., Orintara, S., and Vo, A. (2008). Array CGH data modeling and smoothing in Stationary Wavelet Packet Transform domain. *BMC genomics*, **9 Suppl 2**.

- Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M. R., Li, P., and Oinn, T. (2006). Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, **34**(suppl 2), W729–W732.
- Hupé, P., Stransky, N., Thiery, J.-P., Radvanyi, F., and Barillot, E. (2004). Analysis of array CGH data: from signal ratio to gain and loss of DNA regions. *Bioinformatics*, **20**(18), 3413–3422.
- Infinity Interactive (2011). Moose: A postmodern object system for Perl. <http://moose.iinteractive.com/>.
- Jones, C. (2008). Core LIMS modules in CPAN. <http://search.cpan.org/dist/Microarray/>.
- Jong, K., Marchiori, E., Meijer, G., Vaart, and Ylstra, B. (2004). Breakpoint identification and smoothing of array comparative genomic hybridization data. *Bioinformatics*, **20**(18), 3636–3637.
- Jong, K., Marchiori, E., van der Vaart, A., Chin, S.-F. F., Carvalho, B., Tijssen, M., Eijk, P. P., van den Ijssel, P., Grabsch, H., Quirke, P., Oudejans, J. J., Meijer, G. A., Caldas, C., and Ylstra, B. (2007). Cross-platform array comparative genomic hybridization meta-analysis separates hematopoietic and mesenchymal from epithelial tumors. *Oncogene*, **26**(10), 1499–1506.
- Kallioniemi, A. (2008). CGH microarrays and cancer. *Current Opinion in Biotechnology*, **19**(1), 36–40.
- Kallioniemi, A., Kallioniemi, O. P., Sudar, D., Rutovitz, D., Gray, J. W., Waldman, F., and Pinkel, D. (1992). Comparative genomic hybridization for molecular cytogenetic analysis of solid tumors. *Science*, **258**(5083), 818–821.
- Kane, D., Hohman, M., Cerami, E., McCormick, M., Kuhlman, K., and Byrd, J. (2006). Agile methods in biomedical software development: a multi-site experience report. *BMC Bioinformatics*, **7**(1), 273+.

- Kasprzyk, A., Keefe, D., Smedley, D., London, D., Spooner, W., Melsopp, C., Hammond, M., Rocca-Serra, P., Cox, T., and Birney, E. (2004). EnsMart: A Generic System for Fast and Flexible Access to Biological Data. *Genome Research*, **14**(1), 160–169.
- Kent, W. (1983). A simple guide to five normal forms in relational database theory. *Commun. ACM*, **26**(2), 120–125.
- Khojasteh, M., Lam, W. L., Ward, R. K., and MacAulay, C. (2005). A stepwise framework for the normalization of array CGH data. *BMC bioinformatics*, **6**(1), 274+.
- Kjaerbye-Thygesen, A., Frederiksen, K., Høgdall, E. V., Glud, E., Christensen, L., Høgdall, C. K., Blaakaer, J., and Kjaer, S. K. (2006). Smoking and overweight: negative prognostic factors in stage III epithelial ovarian cancer. *Cancer epidemiology, biomarkers & prevention : a publication of the American Association for Cancer Research, cosponsored by the American Society of Preventive Oncology*, **15**(4), 798–803.
- Klijn, C., Holstege, H., Ridder, J., Liu, X., Reinders, M., Jonkers, J., and Wessels, L. (2008). Identification of cancer genes using a statistical framework for multiexperiment analysis of nondiscretized array cgh data. *Nucl. Acids Res.*, **36**(2), gkm1143+.
- Krzywinski, M., Bosdet, I., Smailus, D., Chiu, R., Mathewson, C., Wye, N., Barber, S., Brown-John, M., Chan, S., Chand, S., Cloutier, A., Girn, N., Lee, D., Masson, A., Mayo, M., Olson, T., Pandoh, P., Prabhu, A.-L. L., Schoenmakers, E., Tsai, M., Albertson, D., Lam, W., Choy, C.-O. O., Osoegawa, K., Zhao, S., de Jong, P. J., Schein, J., Jones, S., and Marra, M. A. (2004). A set of BAC clones spanning the human genome. *Nucleic acids research*, **32**(12), 3651–3660.
- Lai, W. R., Johnson, M. D., Kucherlapati, R., and Park, P. J. (2005). Comparative analysis of algorithms for identifying amplifications and deletions in array cgh data. *Bioinformatics (Oxford, England)*, **21**(19), 3763–3770.

- Lang, D. T. (2007). The R/Splus Perl interface. <http://www.omegahat.org/RSPerl/>.
- Lapointe, J., Li, C., Giacomini, C. P., Salari, K., Huang, S., Wang, P., Ferrari, M., Hernandez-Boussard, T., Brooks, J. D., and Pollack, J. R. (2007). Genomic Profiling Reveals Alternative Genetic Pathways of Prostate Tumorigenesis. *Cancer Research*, **67**(18), 8504–8510.
- Larrañaga, P., Calvo, B., Santana, R., Bielza, C., Galdiano, J., Inza, I. n., Lozano, J. A., Armañanzas, R., Santafé, G., Pérez, A., and Robles, V. (2006). Machine learning in bioinformatics. *Briefings in Bioinformatics*, **7**(1), 86–112.
- Lawrenson, K. and Gayther, S. A. (2009). Ovarian cancer: A clinical challenge that needs some basic answers. *PLoS Med*, **6**(2), e1000025.
- Levi, F. (1999). Cancer mortality in europe, 1990-1994, and an overview of trends from 1955 to 1994. *European Journal of Cancer*, **35**(10), 1477–1516.
- Li, T., Zhang, C., and Ogihara, M. (2004). A comparative study of feature selection and multiclass classification methods for tissue classification based on gene expression. *Bioinformatics*, **20**(15), 2429–2437.
- Liu, J., Mohammed, J., Carter, J., Ranka, S., Kahveci, T., and Baudis, M. (2006). Distance-based clustering of CGH data. *Bioinformatics*, **22**(16), 1971–1978.
- Lockwood, W. W., Chari, R., Chi, B., and Lam, W. L. (2006). Recent advances in array comparative genomic hybridization technologies and their applications in human genetics. *European journal of human genetics : EJHG*, **14**(2), 139–148.
- Marioni, J. C., Thorne, N. P., and Tavaré, S. (2006). BioHMM: a heterogeneous hidden Markov model for segmenting array CGH data. *Bioinformatics*, **22**(9), 1144–1146.
- MATLAB (2012). *version 7.14.0 (R2012a)*. The MathWorks Inc., Natick, Massachusetts.
- Morris, J., Gayther, S., Jacobs, I., and Jones, C. (2008a). A Perl toolkit for LIMS development. *Source Code for Biology and Medicine*, **3**(1), 4+.

- Morris, J. A., Gayther, S. A., Jacobs, I. J., and Jones, C. (2008b). A suite of perl modules for handling microarray data. *Bioinformatics*, **24**(8), 1102–1103.
- Myers, C. L., Dunham, M. J., Kung, S. Y., and Troyanskaya, O. G. (2004). Accurate detection of aneuploidies in array CGH and gene expression microarray data. *Bioinformatics*, **20**(18), 3533–3543.
- Neuvial, P., Hupe, P., Brito, I., Liva, S., Manie, E., Brennetot, C., Radvanyi, F., Aurias, A., and Barillot, E. (2006). Spatial normalization of array-CGH data. *BMC Bioinformatics*, **7**(1), 264+.
- Noble, W. S. (2009). A quick guide to organizing computational biology projects. *PLoS Comput Biol*, **5**(7), e1000424+.
- Olson, N. (2006). The microarray data analysis process: From raw data to biological significance. *Neurotherapeutics*, **3**, 373–383. 10.1016/j.nurx.2006.05.005.
- ORACLE (2012). Java. <http://www.oracle.com/us/technologies/java/index.html>.
- P., G. M. and Angly, F. (2011). Statisitcs::R. <http://search.cpan.org/~fangly/Statistics-R-0.27/lib/Statistics/R.pm>.
- Palmer, S. R. and Felsing, J. M. (2002). *A Practical Guide to Feature-Driven Development*. Prentice Hall, 1 edition.
- Perl.org (2011). Comprehensive Perl Archive Network. <http://www.cpan.org/>.
- Perl.org (2012). Perl programming language. <http://www.perl.org/>.
- Picard, F., Robin, S., Lavielle, M., Vaisse, C., and Daudin, J. J. (2005). A statistical approach for array CGH data analysis. *BMC Bioinformatics*, **6**(1), 27+.
- Pinkel, D. and Albertson, D. G. (2005). Array comparative genomic hybridization and its applications in cancer. *Nature genetics*, **37** Suppl.
- Pinkel, D., Segraves, R., Sudar, D., Clark, S., Poole, I., Kowbel, D., Collins, C., Kuo, W.-L., Chen, C., Zhai, Y., Dairkee, S. H., Ljung, B.-m., Gray, J. W., and

- Albertson, D. G. (1998). High resolution analysis of DNA copy number variation using comparative genomic hybridization to microarrays. *Nature Genetics*, **20**(2), 207–211.
- Poppendieck, M. and Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional.
- Price, D. R. and Ximbiot (2006). Concurrent versions system. <http://www.nongnu.org/cvs/>.
- Python Software Foundation (2012). Python programming language. <http://www.python.org/>.
- Quackenbush, J. (2002). Microarray data normalization and transformation. *Nature Genetics*, **32 Suppl**, 496–501.
- r-project.org (2012). The R project for statistical computing. <http://www.r-project.org/>.
- Rauch, A., Rüschendorf, F., Huang, J., Trautmann, U., Becker, C., Thiel, C., Jones, K. W., Reis, A., and Nürnberg, P. (2004). Molecular karyotyping using an SNP array for genomewide genotyping. *Journal of Medical Genetics*, **41**(12), 916–922.
- Redon, R., Ishikawa, S., Fitch, K. R., Feuk, L., Perry, G. H., Andrews, T. D., Fiegler, H., Shapero, M. H., Carson, A. R., Chen, W., Cho, E. K. K., Dallaire, S., Freeman, J. L., González, J. R., Gratacòs, M., Huang, J., Kalaitzopoulos, D., Komura, D., MacDonald, J. R., Marshall, C. R., Mei, R., Montgomery, L., Nishimura, K., Okamura, K., Shen, F., Somerville, M. J., Tchinda, J., Valsesia, A., Woodwork, C., Yang, F., Zhang, J., Zerjal, T., Zhang, J., Armengol, L., Conrad, D. F., Estivill, X., Tyler-Smith, C., Carter, N. P., Aburatani, H., Lee, C., Jones, K. W., Scherer, S. W., and Hurles, M. E. (2006). Global variation in copy number in the human genome. *Nature*, **444**(7118), 444–454.
- Riedel, S. (2012). mojolicious. <http://mojolicio.us/>.

- Rios, D., McLaren, W., Chen, Y., Birney, E., Stabenau, A., Flicek, P., and Cunningham, F. (2010). A database and API for variation, dense genotyping and resequencing data. *BMC Bioinformatics*, **11**(1), 238+.
- Rouveirol, C., Stransky, N., Hupé, P., Rosa, P. L. L., Viara, E., Barillot, E., and Radvanyi, F. (2006). Computation of recurrent minimal genomic alterations from array-CGH data. *Bioinformatics (Oxford, England)*, **22**(7), 849–856.
- rubyonrails.org (2008). Active record. <http://ar.rubyonrails.org/>.
- Rueda, O. M. and Diaz-Uriarte, R. (2010). Finding recurrent copy number alteration regions: A review of methods. *Current Bioinformatics*, **5**(1), 1–17.
- Saal, L., Troein, C., Christersson, J. V., Gruvberger, S., Borg, A., and Peterson, C. (2002). BioArray Software Environment (BASE): a platform for comprehensive management and analysis of microarray data. *Genome Biology*, **3**(8).
- Savola, S., Klami, A., Tripathi, A., Niini, T., Serra, M., Picci, P., Kaski, S., Zambelli, D., Scotlandi, K., and Knuutila, S. (2009). Combined use of expression and CGH arrays pinpoints novel candidate genes in Ewing sarcoma family of tumors. *BMC Cancer*, **9**(1), 17+.
- Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum (Series in Agile Software Development)*. Prentice Hall, 1 edition.
- Shah, S. P., Xuan, X., DeLeeuw, R. J., Khojasteh, M., Lam, W. L., Ng, R., and Murphy, K. P. (2006). Integrating copy number polymorphisms into array CGH analysis using a robust HMM. *Bioinformatics*, **22**(14), e431–e439.
- Shah, S. P., Lam, W. L., Ng, R. T., and Murphy, K. P. (2007). Modeling recurrent DNA copy number alterations in array CGH data. *Bioinformatics*, **23**(13), i450–458.
- Shah, S. P., Cheung, K.-J., Johnson, N. A., Alain, G., Gascoyne, R. D., Horsman, D. E., Ng, R. T., and Murphy, K. P. (2009). Model-based clustering of array CGH data. *Bioinformatics*, **25**(12), i30–i38.

- Smyth, G. K. (2004). Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Statistical applications in genetics and molecular biology*, **3**(1).
- Smyth, G. K. (2005). *Limma: linear models for microarray data.*, pages 397–420. Springer, New York.
- Solinas-Toldo, S., Lampel, S., Stilgenbauer, S., Nickolenko, J., Benner, A., Döhner, H., Cremer, T., and Lichter, P. (1997). Matrix-based comparative genomic hybridization: biochips to screen for genomic imbalances. *Genes, chromosomes & cancer*, **20**(4), 399–407.
- Staaf, J., Jonsson, G., Ringner, M., and Christersson, J. V. (2007). Normalization of array-CGH data: influence of copy number imbalances. *BMC Genomics*, **8**(1), 382+.
- Statnikov, A., Wang, L., and Aliferis, C. F. (2008). A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. *BMC bioinformatics*, **9**(1), 319+.
- Stein, L. (2010). The case for cloud computing in genome informatics. *Genome Biology*, **11**(5), 207+.
- Stein, L. D. (2007). Cgi.pm. <http://search.cpan.org/~markstos/CGI.pm-3.59/lib/CGI.pm>.
- Stratton, M. R., Campbell, P. J., and Futreal, P. A. (2009). The cancer genome. *Nature*, **458**(7239), 719–724.
- Tang, F., Chua, C. L., Ho, L. Y., Lim, Y. P., Issac, P., and Krishnan, A. (2005). Wildfire: distributed, Grid-enabled workflow construction and execution. *BMC Bioinformatics*, **6**(1), 69+.
- The Apache Software Foundation (2011). Subversion. <http://subversion.apache.org/>.

- The Cancer Genome Atlas Research Network (2008). Comprehensive genomic characterization defines human glioblastoma genes and core pathways. *Nature*, **455**(7216), 1061–1068.
- The International Cancer Genome Consortium (2010). International network of cancer genome projects. *Nature*, **464**(7291), 993–998.
- Theisen, A. (2008). Microarray-based comparative genomic hybridization (aCGH). *Nature Education* **1**(1).
- Tierney, L., Rossini, A. J., Li, N., and Sevcikova, H. (2012). snow: Simple Network Of Workstations. <http://cran.r-project.org/web/packages/snow/index.html>.
- Tiwari, A. and Sekhar, A. K. T. (2007). Workflow based framework for life science informatics. *Computational Biology and Chemistry*, **31**(5-6), 305–319.
- Trout, M. S. (2012). Dbix::class. <http://search.cpan.org/perldoc?DBIx>
- van de Wiel, M. A., Kim, K. I., Vosse, S. J., van Wieringen, W. N., Wilting, S. M., and Ylstra, B. (2007). CGHcall: calling aberrations for array CGH tumor profiles. *Bioinformatics*, **23**(7), 892–894.
- van de Wiel, M. A., Picard, F., van Wieringen, W. N., and Ylstra, B. (2011). Preprocessing and downstream analysis of microarray DNA copy number profiles. *Briefings in Bioinformatics*, **12**(1), 10–21.
- Van Wieringen, W. N., Van De Wiel, M. A., and Ylstra, B. (2007). Weighted clustering of called array cgh data. *Biostat*, pages kxm048+.
- Vapnik, V. N. (1995). *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA.
- Venkatraman, E. S. and Olshen, A. B. (2007). A faster circular binary segmentation algorithm for the analysis of array CGH data. *Bioinformatics*, **23**(6), 657–663.
- Vermeesch, J. R., Melotte, C., Froyen, G., Van Vooren, S., Dutta, B., Maas, N., Vermeulen, S., Menten, B., Speleman, F., De Moor, B., Van Hummelen, P.,

- Marynen, P., Fryns, J.-P., and Devriendt, K. (2005). Molecular Karyotyping: Array CGH Quality Criteria for Constitutional Genetic Diagnosis. *Journal of Histochemistry & Cytochemistry*, **53**(3), 413–422.
- Vincent, J., Vandiver, A., and Glasser, D. (2012). Jifty. <http://jifty.org/>.
- von Eschenbach, A. C. and Buetow, K. (2007). Cancer informatics vision: caBIG. *Cancer informatics*, **2**, 22–24.
- Wang, P., Kim, Y., Pollack, J., Narasimhan, B., and Tibshirani, R. (2005). A method for calling gains and losses in array CGH data. *Biostatistics*, **6**(1), 45–58.
- Wang, Y. and Wang, S. (2007). A novel stationary wavelet denoising algorithm for array-based DNA Copy Number data. *International journal of bioinformatics research and applications*, **3**(2), 206–222.
- Wang, Y., Makedon, F., and Pearlman, J. (2006). Tumor classification based on DNA copy number aberrations determined using SNP arrays. *Oncology reports*, **15 Spec no.**, 1057–1059.
- Wilhelm, M., Veltman, J. A., Olshen, A. B., Jain, A. N., Moore, D. H., Presti, J. C., Kovacs, G., and Waldman, F. M. (2002). Array-based comparative genomic hybridization for the differential diagnosis of renal cell cancer. *Cancer research*, **62**(4), 957–960.
- Willenbrock, H. and Fridlyand, J. (2005). A comparison study: applying segmentation to array CGH data for downstream analyses. *Bioinformatics (Oxford, England)*, **21**(22), 4084–4091.
- Yang, Y. H., Buckley, M. J., and Speed, T. P. (2001). Analysis of cDNA microarray images. *Briefings in bioinformatics*, **2**(4), 341–349.
- Yang, Y. H., Dudoit, S., Luu, P., Lin, D. M., Peng, V., Ngai, J., and Speed, T. P. (2002). Normalization for cDNA microarray data: a robust composite method addressing single and multiple slide systematic variation. *Nucleic Acids Research*, **30**(4), e15.

Zeggini, E., Scott, L. J., Saxena, R., Voight, B. F., Marchini, J. L., Hu, T., de Bakker, P. I. W., Abecasis, G. R., Almgren, P., Andersen, G., Ardlie, K., Bostrom, K. B., Bergman, R. N., Bonnycastle, L. L., Borch-Johnsen, K., Burt, N. P., Chen, H., Chines, P. S., Daly, M. J., Deodhar, P., Ding, C.-J., Doney, A. S. F., Duren, W. L., Elliott, K. S., Erdos, M. R., Frayling, T. M., Freathy, R. M., Gianniny, L., Grallert, H., Grarup, N., Groves, C. J., Guiducci, C., Hansen, T., Herder, C., Hitman, G. A., Hughes, T. E., Isomaa, B., Jackson, A. U., Jorgensen, T., Kong, A., Kubalanza, K., Kuruvilla, F. G., Kuusisto, J., Langenberg, C., Lango, H., Lauritzen, T., Li, Y., Lindgren, C. M., Lyssenko, V., Marvelle, A. F., Meisinger, C., Midthjell, K., Mohlke, K. L., Morken, M. A., Morris, A. D., Narisu, N., Nilsson, P., Owen, K. R., Palmer, C. N. A., Payne, F., Perry, J. R. B., Pettersen, E., Platou, C., Prokopenko, I., Qi, L., Qin, L., Rayner, N. W., Rees, M., Roix, J. J., Sandbaek, A., Shields, B., Sjogren, M., Steinthorsdottir, V., Stringham, H. M., Swift, A. J., Thorleifsson, G., Thorsteinsdottir, U., Timpson, N. J., Tuomi, T., Tuomilehto, J., Walker, M., Watanabe, R. M., Weedon, M. N., Willer, C. J., Illig, T., Hveem, K., Hu, F. B., Laakso, M., Stefansson, K., Pedersen, O., Wareham, N. J., Barroso, I., Hattersley, A. T., Collins, F. S., Groop, L., McCarthy, M. I., Boehnke, M., and Altshuler, D. (2008). Meta-analysis of genome-wide association data and large-scale replication identifies additional susceptibility loci for type 2 diabetes. *Nature Genetics*, **40**(5), 638–645.

Appendices

.1 Source code appendix

This appendix lists the name and file type of each source code files included on the additional CD by chapter.

.1.1 Chapter 2

File name	File Type
Base.pm	Perl module
Controller.pm	Perl module
Interface.pm	Perl module
Util.pm	Perl Module
lims_index.cgi	Perl CGI script
lims_login.cgi	Perl CGI script
admin.cgi	Perl CGI script
collection_admin.cgi	Perl CGI script
filter_admin.cgi	Perl CGI script
fluorochrome_admin.cgi	Perl CGI script
laser_admin.cgi	Perl CGI script
protocol_admin.cgi	Perl CGI script
specimen_admin.cgi	Perl CGI script
tumour_admin.cgi	Perl CGI script
user_admin.cgi	Perl CGI script
patient.cgi	Perl CGI script
specimen.cgi	Perl CGI script
ext_specimen.cgi	Perl CGI script
tumour.cgi	Perl CGI script
labelling.cgi	Perl CGI script
file_download.cgi	Perl CGI script

.1.2 Chapter 3

File name	File Type
<code>process_data.pl</code>	Perl script
<code>process_data_test.pl</code>	Perl test script
<code>ArrayPipeLine.pm</code>	Perl module
<code>pipeline_analysis.t</code>	Perl test script
<code>PipelineAnalysis.pm</code>	Perl module
<code>PipelineStep.pm</code>	Perl module
<code>LowessAnalysis.pm</code>	Perl module
<code>ManorAnalysis.pm</code>	Perl module
<code>CGHcallAnalysis.pm</code>	Perl module
<code>Pipeline_Data.pm</code>	Perl module
<code>pipeline_data_test.pl</code>	Perl test script
<code>Image.pm</code>	Perl module
<code>QC_Plots.pm</code>	Perl module
<code>CGH_Plot.pm</code>	Perl module
<code>plot_generator.cgi</code>	Perl CGI script
<code>plot_view.cgi</code>	Perl CGI script
<code>image_viewer.cgi</code>	Perl CGI script
<code>manor_view.cgi</code>	Perl CGI script
<code>manor.cgi</code>	Perl CGI script
<code>MANOR_normalisation.r</code>	R script
<code>DNAcopy_segmentation.r</code>	R script
<code>lowess_normalisation.r</code>	R script

.1.3 Chapter 4

File name	File Type
<code>aCGH_analysis_functions.r</code>	R library
<code>recurrent_region_analysis.r</code>	R script
<code>recurrent_region_input_parser.pl</code>	Perl script
<code>recurrent_region_input_parser_test.pl</code>	Perl script
<code>region_selection.r</code>	R script
<code>region_validation.r</code>	R script
