Variant calling from next generation sequencing data and information management

Eliza Ralph
Master's Thesis
Master's Degree Programme in Bioinformatics
Department of Information Technology
University of Turku
November 2014

UNIVERSITY OF TURKU

Department of Information Technology

RALPH, ELIZA: Variant Calling from Next Generation Sequencing Data and Information Management

Master's Thesis, 63 p., 24 appendix pages

Master's Degree Programme in Bioinformatics

November 2014

The aim of the thesis is to develop a solution for the data management of heterogeneous data sets based on the Glanville fritillary butterfly (Melitaea cinxia), a key model species in metapopulation ecology subjected to large-scale sequencing. Specific analysis needs arose from several NGS sequencing projects in the Metapopulation Research Group (MRG), University of Helsinki, requiring a method to store and having easy access to non-public datasets.

Variant calling is the process of making the distinction between biological variants and sequencing errors from Next Generation Sequencing (NGS) data. Variants discovered from calling include SNPs (single nucleotide polymorphisms) and indels (insertion/deletion mutations).

Methods for generating called variants from NGS experiments and variation analysis tools were reviewed and an example variant calling pipeline presented. The end result of variant calling produces a VCF (*Variant Call Format*) file, a tab-delimited file format for storage of variation data.

Technical methods for extraction, storage and analysis of data were examined. Different database paradigms, NoSQL versus relational database were compared. The ETL (*Extract, Transform Load*) process for extracting the data into a database was developed using SQL Server Integration Services (SSIS). The relational database model was designed in SQL Server 2014. The web-based front-end for querying data was developed in ASP.NET MVC 5.

The solution enables the study of variation between and within populations and landscapes, select interesting SNPs and aid in the design of genotyping experiments. The primary aim is to show locations of variation in genomic scaffolds. Due to use of a standardised VCF format, the application has the possibility to be applied to studying variation in other organisms.

Keywords: Glanville fritillary butterfly, *Melitaea cinxia*, NGS, Next generation sequencing, variant calling, SNP calling, variation, sequencing, metapopulation, database, SQL Server

Contents

1 Introduction

Genetic variation is the variation in the genome among individuals and populations as a result of genetic mutation. At a molecular level, variation refers to the differences in nucleotide bases in DNA. Genetic variation in biological populations plays an important role in local adaptation of species to changing environments, and hence to saving biodiversity.

Variant calling is the process of finding biological variants from Next Generation Sequencing (NGS) data. Variants discovered from calling include SNPs (single nucleotide polymorphisms, pronounced "*snips*") and indels (insertion/deletion mutations). SNPs are the most commonly occurring type of genetic variation.

NGS refers to the current range of high-throughput sequencing (HTS) technologies. HTS was given rise to by the sheer quantities of sequencing data produced, driven by the lowered cost of modern sequencing platforms compared to earlier sequencing methods.

One of the many applications of NGS is variation analysis. Discovering variation from NGS data poses challenges that are algorithmically and computationally complex. Currently many tools to aid variant analysis exist yet much work still remains. The main challenge in variant analysis is identifying true variation versus sequencing errors (Dalca & Brudno, 2010).

The aim of the thesis is to develop a solution for the data management of heterogeneous data sets based on the Glanville fritillary butterfly (*Melitaea cinxia*). The Glanville fritillary is a key model species in metapopulation ecology and is one of the first non-model species of which the whole genome sequence is now available (Ahola et al., 2014). The data used in the database originates from the RNA-seq gene expression study from two Glanville fritillary populations. The data was prepared by the Metapopulation Research Group (MRG), Department of Biosciences, University of Helsinki.

Data is aggregated from VCF (*Variant Call Format*) and metadata including population data. The VCF format contains resultant SNPs and indels from variant calling with a level of confidence associated with the call specified by a Phred-based quality score (Danecek et al., 2011).

The intended audience is for researchers and lab staff. The application allows researchers to easily query data without expert knowledge required of the system. The application is developed for the web for ease of access and to avoid the need to install software.

The solution enables the study of variation between and within populations and landscapes, select interesting SNPs and to aid in the design of genotyping experiments. The primary aim is to show locations of variation in scaffolds. Publicly available tools were found not to address the particular needs of the data gathered.

Variation analysis methods and the principals of variant calling were reviewed. Different database paradigms, NoSQL versus relational database were compared. A relational database model was designed in SQL Server 2014. The web-based front-end for querying data was developed in ASP.NET MVC 5.

2 Variation in Ecological Genomics

This chapter reviews the role of variation studies in the discipline of ecological genomics and introduces important terminology in metapopulation dynamics to understand the discipline the data for this study is sourced from. The motivations for the study of the non-model organism, Glanville fritillary butterfly is introduced.

2.1 Ecological and population genomics

Ecological genomics is the study of genomics in natural populations. Population ecology (a subfield of ecology) encompasses the evolutionary and ecological processes affecting populations in order to gain understanding on the abundance and interactions of species (Levin, 2009).

Molecular ecology applies techniques from several specialised fields that use molecular genetics techniques focusing on biomolecules (e.g. DNA, RNA, protein) to answer ecological and evolutionary questions (Andrew et al., 2013).

Due to the sheer abundance of data produced by NGS, ecological genomics studies has seen a shift from model organisms in a lab setting to non-model organisms in natural populations. This has resulted in the opening up of many new areas in ecological genomics studies with the ability to obtain definitive answers to common ecological questions at a magnitude not previously seen before (Ekblom & Galindo, 2011).

The discipline of population genomics contains concepts that can be used interchangeably with population and molecular ecology encompassing the distribution of genetic variation between populations influenced by evolutionary processes: Selection, inbreeding, genetic drift, gene flow, mutation and recombination (Levin, 2009). These factors in force within populations may ultimately lead to evolution (Andrews, 2010). Vital to studies in population genomics is to differentiate between separating effects of demographic history and natural selection (Nielsen et al., 2007).

Selection *contributes to the removal of disadvantageous genes from populations (Ahluwalia, 2009).*

Inbreeding *is a form on non-random mating of individuals that are closely related genetically (Ahluwalia, 2009).*

Genetic drift *is the random unexpected changes that occur in gene frequency (Ahluwalia, 2009).*

Gene flow *describes the genetic exchange between populations.*

Mutation *is a source of genetic variation.*

Recombination *is the formation of new combinations of genetic material.*

The basis of natural selection involves the survivability of given traits given by reproductive success (i.e. the *fitness*) in a natural population. Ultimately, selection leads to evolution. For evolution to occur, natural selection requires these three variables: Genetic variation within the population, heredity and differential reproduction (Levin, 2009).

Fitness *according to Darwinian natural selection, is a measure of an organism's capacity to survive and reproduce (Levin, 2009).*

NGS data in ecological and population genomics research has two major application areas: *Transcriptomic* or *genomic* (Ekblom & Galindo, 2011). Transcriptomics is the study of the transcriptome i.e. the complete set of RNA expressed in an organism. As the focus of this thesis is variant calling, the various other applications of NGS data in ecological studies are out of scope.

SNPs and indels may describe variation caused by mutation or alteration that can be observed in the form of a DNA sequence. Discovered polymorphisms can therefore be used to discern differences within and across populations. Associated changes in a gene function and expression as a result of variation may contribute to greater fitness. Changes favoured by natural selection contribute overall to evolutionary change (Norrgard & Schultz, 2008).

NGS data in ecological studies plays a significant role in the large-scale identification and development of genetic markers (Ekblom & Galindo, 2011). A genetic marker is an identifiable segment of DNA that can be used in the identification of tissue, individuals, species or populations. SNPs can be used as genetic markers to answer common questions in population genomics (Morin, Luikart, & Wayne, 2004), for instance, for identifying genes involved in ecologically important phenotypic variation (Ekblom & Galindo, 2011).

2.2 Metapopulation dynamics

A *metapopulation* is a group of spatially separated populations linked via dispersal and *dispersal* simply is the movement of organisms away from their place of birth. Study in metapopulation ecology addresses the ecological, genetic and evolutionary processes that occur in metapopulations.

Metapopulation dynamics incorporates concepts from *landscape ecology* where a landscape is "the human-defined area of the natural terrestrial world" (Levin, 2009). A *habitat patch* is a continuous range of environment conditions under which a species may survive and reproduce

(Gutzwiller, 2002). A *landscape matrix* describes the interconnectivity between patch types that constitute a landscape.

Suitable habitats in which populations can persist, occur in discrete habitat patches forming a greater habitat patch network consisting of local populations that make up a metapopulation. In highly-fragmented landscapes, local populations may be so small they risk extinction that can be compensated by a few dispersing individuals.

The three facets of eco-evolutionary dynamics are defined as "dispersal, colonization and local extinction". These three factors depend on species traits, habitat/landscape characteristics and population state, and largely affect a population's stability. Dispersal due to emigration/immigration of individuals is an influencing factor on population dynamics that contributes to gene flow and genetic variation. Dispersal rates between populations are measured via mathematical models in which c*onnectivity* is a measure of the expected dispersal rates to a particular habitat patch from surrounding populations.

One of the key research areas in evolutionary genomics and metapopulation biology it to study adaptation to local environments. Adaptation refers to the evolution of traits given by genetic variation in a population by the process of natural selection. Populations and metapopulations that have become locally adapted may exhibit higher fitness than other populations (Levin, 2009).

2.3 Non-model organism: Glanville fritillary butterfly

Considered to be a key model species in studies of metapopulation ecology (Hanski, 1999), the Glanville fritillary (*Melitaea cinxia*) is a text-book example of a species with complex meta-population dynamics (Ekblom & Galindo, 2011) subject to large-scale sequencing (Ellegren & Sheldon, 2008). The Glanville fritillary was the first non-model species of which the transcriptome was sequenced and assembled *de novo* (without a reference sequence) using NGS (Vera et al., 2008).

3 Next Generation Sequencing

This chapter reviews at a high level NGS, an important precursor to the variant analysis techniques discussed in this thesis. Significant sequencing projects are reviewed, discussing the important role played in the current state of variant analysis.

3.1 NGS technology

NGS platforms such as the Genome Analyzer (*Illumina*), ABI SOLiD (*Applied Biosystems* now *Life Technologies*) and 454 (*Roche*) are enablers of high throughput variation analysis in the genome of human and other species, including non-model organisms (Dalca & Brudno, 2010).

NGS is characterized by numerous relatively high-quality short sequence stretches called reads, typically 25 – 500 base pairs (*bp*) in length. The maximum read length varies between vendors and platforms.

The Illumina HiSeq 2500 platform is capable of generating up to a maximum of 1000 Gb (Illumina, 2014) in a single run. The sheer quantities of data produced poses challenges that are algorithmically and computationally complex (Nielsen et al., 2007).

The main applications of NGS include resequencing and seq-based techniques including RNA-seq and ChIP-Seq, and *de novo* sequencing (Dalca & Brudno, 2010). NGS sequencing experiments can sequence the whole genome or targeted regions (e.g. the exome) (Altmann et al., 2012).

Illumina HiSeq 2500 *is the current NGS platform from Illumina for large scale genomics* (Illumina, 2014).

Gb (gigabase) *refers to 1 million nucleotide bases.*

Reference genome *is a consensus nucleotide sequence curated by scientists that digitally represents all the DNA of an organism assembled from the sequenced DNA of one or many individuals. The reference genome assembled from many individuals is a haploid representation which does not provide an accurate representation of any given individual. (Dalca & Brudno, 2010).*

Resequencing *is an NGS method for variation analysis in relation to a reference genome (Flicek & Birney, 2010). Resequencing is typically used in order to differentiate individuals within and between families and populations (Nakamura et al., 2011).*

RNA-seq *(RNA sequencing) is an NGS method of RNA analysis through the sequencing of cDNA (Corney, 2012).*

ChIP-seq *is an NGS method for studying protein interactions and histone modifications combining chromatin immunoprecipitation (ChIP) (Head et al., 2014).*

de novo *sequencing is the sequencing of novel genomes.*

Library *refers to a set of nucleic acid fragments that has been processed and is ready for sequencing (Head et al., 2014).*

The precursor to sequencing is sample preparation where DNA or RNA is extracted from biological material are fragmented followed by template preparation where fragments are converted into the library and clonally amplified. The methods for template preparation are vendor specific (Harbers & Kahl, 2012). An *adapter* is an oligonucleotide designed to interact with the NGS platform added to the ends of the fragment during template preparation.

Sequencing experiments can produce single-end or paired end reads. Single-end read DNA sequencing is a simple approach to sequencing DNA from a single end of a fragment. Paired-end reads are a methodology used in DNA/RNA sequencing where both ends of fragments are sequenced coupled with distance information. The main benefit of paired-end reads is increased specificity when aligning to the genome and discovering structural variants (SVs).

Sequencing is either determined with *sequencing-by-synthesis* (SBS) or *sequencing-by-ligation* (SBL). Sequencing-by-synthesis is a base-space sequencing method that can be further categorised into terminal based method (e.g. the Illumina platform) and pyrosequencing (e.g. the Roche 454 platform). The ABI SOLiD platform is a colour-space sequencing-by-ligation method (Altmann et al., 2012). Discussing the exact mechanistic details and chemistry is out of the scope of this thesis although it is worth considering that NGS platforms differ in terms of their hardware, chemistry and optical sensors in order to elicit data from a sequencing experiment. These are important factors when considering the different types of errors associated with sequencing experiments (Ledergerber & Dessimoz, 2011). Errors commonly arise from imperfection in the chemistry and/or sensors and are discussed in further detail in section 4.2.5.

3.2 Significant sequencing projects

Genome projects that have had a major impact in genomics include the *1000 Genomes Project* and the *International HapMap Project*. Although both projects focus on human genetics, both projects have had a substantial impact in genomics research.

With the purpose for studying disease in humans, the **International HapMap project** is a large scale genotyping project providing a public catalogue of human genetic variation. Based on the premise that haplotypes are generally shared between populations but frequencies differ, the

Genotyping *is the process of determining the genotype i.e. the genetic constitution of an individual (Coriell Institute for Medical Research, 2014).*

Haplotype *blocks are regions of genomic DNA containing a series of SNPs associated by LD (linkage disequilibrium). (Harbers & Kahl, 2012).*

HapMap project provides a shortcut to whole genome sequencing ("What Is the HapMap," n.d.). Completed in approximately three years, the HapMap project characterised 1.6 million SNPs with allele frequencies in 11 human populations from Africa, Asia, Europe and America (Altshuler et al., 2010). HapMap data was one of the data sources utilised by the 1000 Genomes Project discussed further below (Nielsen et al., 2007). (DePristo et al., 2011)

The **1000 Genomes Project** launched in 2008, is a community distributed catalogue of human genetic variation serving as a prime example of a successful application of how low coverage sequencing utilising NGS technologies became the cornerstone of many current bioinformatics, analysis and data distribution methods for genomic data. The aim of the project was to discover sequence variants with minor allele frequencies in at least 1% of the populations studied ("About the 1000 Genomes Project," n.d.). As the name suggests, the initial plan was to sequence 1000 individuals with the number having grown to 2577 at the time of writing.

VCF is an example NGS method developed for the 1000 Genomes Project that has become a mainstream technique in variant analysis. VCF is the adopted format of NCBI's mutation database *dbSNP* (Danecek et al., 2011). LD (*linkage disequilibrium*) and recalibration are other example NGS methods utilised by the 1000 Genomes Project that have been adopted in other variation analysis projects (Abecasis et al., 2010).

Variant Call Format (VCF) *is a tab delimited text format which encodes DNA polymorphism data (Samtools, 2013).*

Linkage disequilibrium (LD) *is the non-random association of alleles at two or more loci and is commonly used as a measure for correlations between linked loci (Peng et al., 2012).*

Recalibration *refers to the recalibration of quality scores in a variant calling pipeline (DePristo et al., 2011).*

The three pilot projects identified approximately 5 million SNPs, 1.5 million indels and 22,000 SVs. Of the three pilot projects; 55, 57 and 61% of variants were novel. The release of the data was the single biggest contribution to *variation* studies (Casals, Idaghdour, Hussin, & Awadalla, 2012).

4 Variant Analysis

In the first part of the chapter the biological background for variant analysis is reviewed and a deeper look into the definition of a SNP is covered. An example, well-studied gene Phosphoglucose isomerase (*Pgi)* provides context on variation studies into the Glanville fritillary. In the second part of the chapter, a typical variant calling pipeline for DNA-seq (DNA sequencing) is presented. Finally, a review of existing SNP databases and tools used in analysing variant data is presented.

4.1 Sources of genetic variation

In section 2.1 factors within populations that lead to genetic variation and ultimately evolution were raised (selection, inbreeding, genetic drift, gene flow, mutation and recombination). In this section, the biological effects of mutation and recombination are examined.

4.1.1 Gene structure

Composed of segments of DNA molecules, genes (Figure 4.1) are considered to be the basic units of inheritance (Pearson, 2006). DNA which comprises genes themselves contains intergenic regions of DNA sequence preceding genes that is proven to have regulatory functions. Eukaryotic protein-coding genes vary in size and organization thus not conforming to a typical structure. However, there are some several conserved features. The boundaries of the protein coding gene are defined by the point where transcription begins and ends. The core of the gene is the protein coding region. The coding region begins with an initiation codon and ends with a termination codon (Twyman, 2003). At either end of the gene on each side of the coding sequence is an untranslated region (UTR) known as the *5' UTR* and *3' UTR* ("' " pronounced as *prime*). The coding region and UTR regions may be interrupted by introns (non-coding sections within the coding region). Non-coding regions and UTRs contain regulatory regions including a promoter located upstream at the 5' end and one or more enhancers located upstream or downstream of the gene it regulates (Clark, 2009).

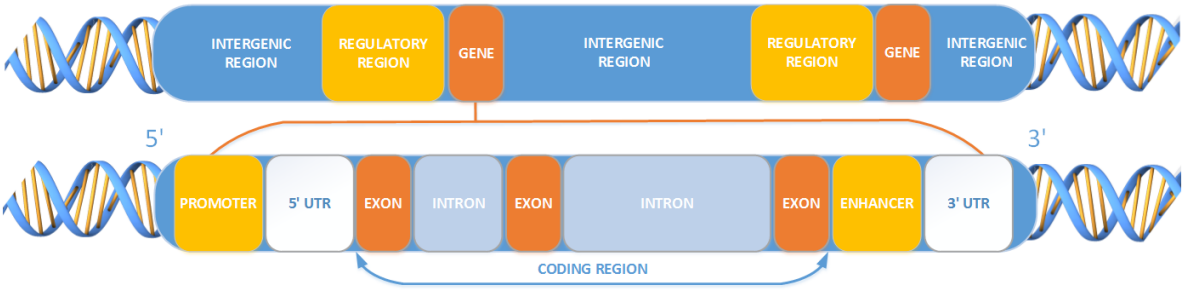


Figure 4.1 - Simplified representation of a eukaryotic protein-coding gene. The figure exhibits how a DNA strand contains many genes and their corresponding regulatory regions preceded by intergenic regions. The gene is further broken down to describe at a high level the genic structure of a eukaryotic protein coding gene from the 5' to 3' end. DNA double helix component of image licenced under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 (http://creativecommons.org/licenses/by/3.0/) (Hedberg, 2013).

4.1.2 Genetic variation

Genetic variation is the genetic differences in alleles of genes between individuals. Single Nucleotide Polymorphisms (SNPs), frequently pronounced as "snips" (Figure 4.2) are the most commonly occurring type of genetic variation where one nucleotide base (A, C, T or G) is substituted for another. *Polymorphisms* that effect a single base can be described as *insertion* or *deletion* (indels) or *substitution* mutations and are otherwise known as *point mutations* (Clark 2009).

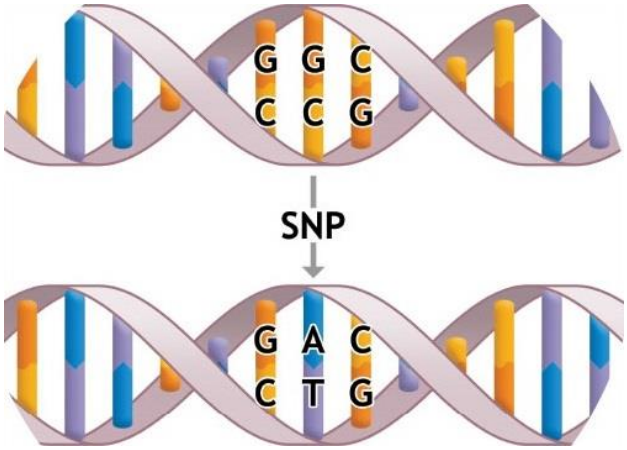


Figure 4.2 - A SNP is a single nucleotide change at a single base-pair location in DNA. Image courtesy of http://www.ibbl.lu/. Rights of reproduction are reserved and limited (IBBL, n.d.).

4.1.3 Classes of SNPs

SNPs are typically classified according to their position in the genome and can be identified by two variant classes: *Coding* and *non-coding* (Charlesworth, 2010). Located in the coding region, *coding SNPs* (cSNPs) can either be synonymous or nonsynonymous (Ahluwalia, 2009). Synonymous (or

silent) *SNPs* have different alleles that code for the same amino acid thus having no effect on amino acid sequences. Nonsynonymous SNPs have different alleles that code for different amino acid sequences resulting in *missense* or *nonsense* mutations ("SNP Class Definitions," 2005). In a missense mutation, a codon change of one amino acid to another occurs whereas in a nonsense mutation the codon for an amino acid is mutated to give a stop codon.

4.1.4 DNA repair and mutation

Mutation is seen to be the ultimate source of genetic variation characterized by a chemical change altering the sequence of nucleotide bases in DNA. Organisms contain a variety of mechanisms to repair DNA in the event of damage (Clark, 2009). Sources of damage to DNA include damage by agents (e.g. chemical or radiation) or replication infidelity (Carlin, 2011) with damage resulting in breaks in the chromosome or cell death due to blocks in replication (Clark, 2009). Unrepaired DNA or failure in the replication apparatus due to the inclusion of incorrect bases may result in the development of mutations that may be neutral, deleterious and in some cases advantageous, where new alleles can be favoured by selection (Andrews, 2010). Inheritance of mutation to offspring occurs only if mutations occurring in the germ-cells are passed on during sexual reproduction, otherwise mutations occurring in somatic cells are only passed on to descendants of those cells (Clark, 2009).

4.1.5 Recombination

In eukaryotes, variation introduced by reproduction is described by the phenomena known as *recombination* whereby genetic material are shuffled between chromosomes, contributing towards greater genetic diversity in offspring and benefiting evolutionary selection (Clark, 2009). At a molecular level, *crossover* describes the phenomena where DNA molecules are broken down and re-joined between homologous chromosomes in order for recombination to occur (Clark, 2009).

4.1.6 Example gene: *Pgi*

Phosphoglucose isomerase (*Pgi)* gene, encoding for phosphoglucose isomerase, is a well-studied, highly-polymorphic gene involved in the glucose metabolism of many species. The *Pgi* gene catalyses the conversion of glucose-6-phosphate to fructose-6-phosphate in the first phase of glycolysis (Ellegren & Sheldon, 2008).

Pgi serves as a prime example of how genetic variation at a fitness-related locus affects population dynamics and growth of the Glanville fritillary populations. Studies indicate a strong association between one *Pgi* SNP and a range of life history traits (flight metabolic rate, dispersal rate, body temperature at flight, clutch size, lifespan and pupal weight) with the AC heterozygote being typically more favourable than AA homozygotes (Hanski, 2011).

4.2 Variant Calling Pipeline

The extraction of SNPs from raw genetic material (DNA or RNA) involves many processing steps constituting a SNP/genotype calling pipeline built from a diverse set of tools (Altmann et al., 2012). Figure 4.3 illustrates a typical SNP/genotype calling pipeline where the first two (2) phases in the diagram are common to nearly all HTS applications and the remaining three (3) phases specific to SNP/Genotype calling (Altmann et al., 2012). The various phases are discussed in further detail over the next few sections.

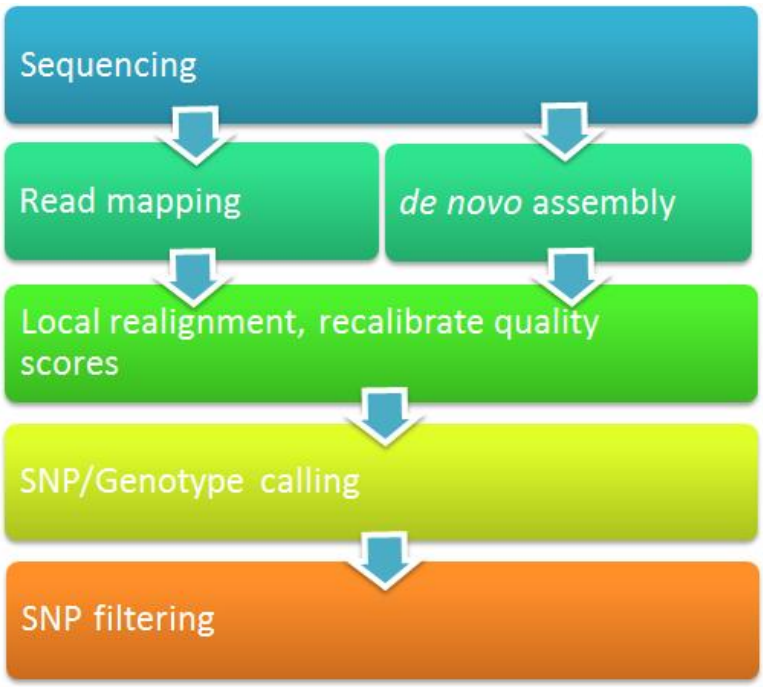


Figure 4.3 – SNP/Genotype calling pipeline. Steps for converting raw sequence data to SNP calls. Sequencing incorporates base-calling. Read mapping is performed if a reference genome exists, otherwise *de novo* assembly is performed. Prior to SNP/Genotype calling pre-processing improves calling accuracy. Realignment better aligns e.g. small indels. Quality scores are adjusted. Depending on the type of study, SNP or genotype calling is performed. Typically statistical approaches are used to infer SNPs/genotypes from cleansed mapped read data. Filtering reduces the number of false-positive SNP calls. (Dalca & Brudno, 2010).

During sequencing, raw read data is transformed into a standardised format with well-calibrated base-call estimates (DePristo et al., 2011).

Read mapping algorithms align reads from the sequencing phase against a reference genome and determines the position of each read (DePristo et al., 2011). A further discussion into read

mapping algorithms is discussed in further in section 4.2.2 with particular attention paid to the BWA algorithm, a popular algorithm used in resequencing experiments (H. Li & Durbin, 2009).

Pre-processing takes place after read mapping to improve the function of variant calling algorithms (Altmann et al., 2012). Pre-processed files are subjected to variant calling algorithms by applying statistical methods to identify sites containing biological variants (DePristo et al., 2011). The resultant output from variation analysis experiments is typically a list of variants between the sequenced genome and the reference genome (Dalca & Brudno, 2010).

Section 4.2.5 introduces the common types of errors in NGS variant analysis experiments, often attributed to by imperfections in chemistry and/or sensors (Ledergerber & Dessimoz, 2011) with a discussion on methods for improving accuracy.

In contrast, the key difference of a typical RNA-seq variant calling pipeline (as opposed to DNA-seq) is handling splice junctions correctly which requires specific read mapping algorithms and pre-processing procedures (H. Li & Homer, 2010).

4.2.1 Sequencing

Sequencing determines the exact order of nucleotide bases from DNA. Base-calling forms part of the sequencing process where software analysis of sensor data is used to predict individual nucleotide bases (Ledergerber & Dessimoz, 2011). Most platforms provide the reads from base-calling directly to a flat file format such as the standardised *FASTQ* format (e.g. Illumina platform) or provide data conversion tools (Altmann et al., 2012). Initial estimations of quality is conducted during this phase with an associated quality score stored in the FASTQ file often provided by base-calling software of the manufacturer (Altmann et al., 2012). Base-calling methods vary between vendors, each with their own platform-specific biases (Ledergerber & Dessimoz, 2011).

FASTQ *is a text-based storage format for nucleotide sequences with associated quality scores.*

4.2.2 Read mapping

Read mapping (otherwise known as *short-read sequence alignment*) is part of the variant calling pipeline where short reads from an individual's sequence are mapped or *aligned* to the reference genome (Nielsen, Paul, Albrechtsen, & Song, 2011) and their position within the reference genome is determined (Dalca & Brudno, 2010). This implicitly assumes that there is a reference sequence. In cases where a sufficient reference genome does not exist or mapping of reads is not enough to

identify segments, *de novo* sequence assembly methods are applied. Read mapping algorithms differ from traditional alignment algorithms, specifically catering for short reads, and are considered to be one of the most computationally intensive phases of the pipeline (H. Li, Ruan, & Durbin, 2008).

Mapped reads are output to SAM/BAM (*sequence alignment/map* and the more performant binary counterpart *binary alignment/map*) formats. SAM is a standardized, tab delimited text format for encoding nucleotide sequence alignments; serving an intermediary step in the SNP calling pipeline where the results are stored for variant calling and other downstream analyses (H. Li & Durbin, 2009).

The accuracy of read mapping plays a crucial role in variant detection. Incorrectly mapped reads can lead to errors in variant calling. It is imperative for read mapping algorithms to deal with sequencing errors and potentially real differences (e.g. point mutations and indels) (Nielsen et al., 2011).

Read mapping algorithms

A majority of existing read mapping algorithms fall into two classes: Hashing or BWT (Burrows–Wheeler transform) algorithms. Hash-table implementations create a hash either from the reference genome or the set of sequencing reads (Flicek & Birney, 2010). Examples of common read mapping programs include BWT-based algorithms BOWTIE/BOWTIE2, BWA and SOAP2 and hash-based algorithms MAQ, SHRiMP2 and BFAST (not a comprehensive list) (H. Li & Durbin, 2009), (Altmann et al., 2012).

BWT-based aligners are fast, memory-efficient and useful for aligning repetitive reads, tending to be less sensitive than hash-based algorithms (Nielsen et al., 2011). Li & Durban (2009) suggests that BWA is approximately 10-20 times faster than the hash-based algorithm MAQ whilst achieving similar accuracy. *BowTie2* is another commonly accepted standard in alignment algorithms. Unlike MAQ; BWA and BowTie2 support mismatches, gaps and indel alignment and CPU parallelism support (Altmann et al., 2012).

BWT-based algorithms are best known for their implementation of the *bzip2* compression standard (Seward, n.d.). BWT-based aligners apply indexing to the reference genome to support search with fewer memory resources and are typically chosen when performance is favoured (H. Li

& Durbin, 2009). BWT methods are typically built with the *FM index* data structure (Ferragina & Venturini, 2005).

When selecting an alignment algorithm, parameter settings should be considered. For example, perfect matches will not allow for the identification of SNPs but a balance is required between correctly identifying SNPs and false positives (Altmann et al., 2012).

The Burrows-Wheeler Aligner (BWA)

The *Burrows-Wheeler Aligner* (BWA) algorithm is based on the BWT-based FM-index data structure (Nielsen et al., 2011).

The FM index introduced the idea that a *suffix array* created from a BWT sequence is more efficient than the original sequence (Figure 4.1). From computer science, a suffix array is a sorted array of all suffixes of a string. A suffix trie (or simply a *trie*) is a data structure that stores string suffixes to perform fast string matching (H. Li & Homer, 2010).

Creating the underlying data structure of the FM-index is a two-step process. Firstly, BWT is applied to the trie-based FM index data structure to create a suffix array from the original sequence (Ferragina & Venturini, 2005). A single reference to multiple sequence instances is created by collapsing identical copies into a single path in the trie alignment. Secondly, indexing is applied (H. Li & Homer, 2010). The benefits of the FM-index are supporting rapid sub-sequence search whilst maintaining a size smaller or equal to the input genome (Flicek & Birney, 2010).

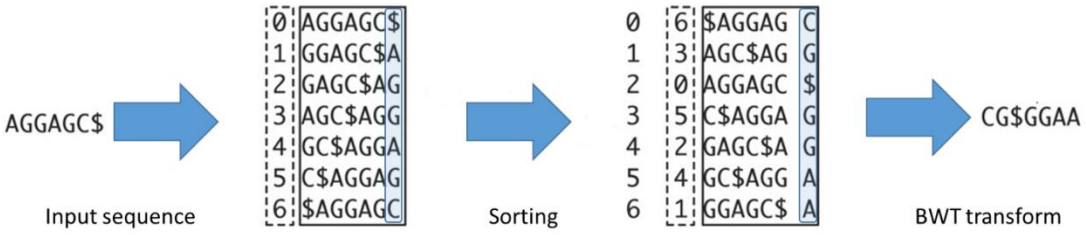


Figure 4.1 - Constructing the suffix array and the BWT transform of AGGAGC where $ denotes the end of a string. The input sequence is circular shifted to represent 7 sequences which are then sorted lexicographically. After sorting, the positions of the first symbols form the suffix array (6, 3, 0, 5, 2, 4, 1) and the concatenation of the last symbols of the circulated strings gives the BWT string. Adapted by permission from Oxford University Press: BRIEFINGS IN BIOINFORMATICS (H. Li & Homer), copyright (2010).

Backward search is an alternative method to exact string matching which samples distinct substrings from the genome which in effect mimics a top-down traversal of the prefix trie,

resulting in a smaller memory footprint without requiring that the trie be explicitly stored in memory (H. Li & Durbin, 2009).

Selecting an algorithm may depend on sequence length and level of similarity. *BWA-MEM* is suited for sequences 70 bp or longer, *BWA-backtrack* is most suited for smaller sequences and *BWA-SW* has better sensitivity when alignment gaps are frequent ("Burrows-Wheeler Aligner," n.d.).

Recent advancements

Recent advancements in read mapping algorithms include a *random permutation* based-approach with improved speed and sensitivity (Lederman, 2013) and the *sub-read aligner strategy* utilising the 'seed-and-vote' paradigm. When coupled with traditional read mapping algorithms the sub-read strategy uses overlapping reads (*sub-reads*) to vote on the optimal location of the read which is less computationally expensive for large datasets and supports longer read lengths (Liao, Smyth, & Shi, 2013).

Read mapping of RNA-seq data

The first step in analysis of transcriptome sequencing (RNA-seq) experiments is to map short reads to a reference genome or perform *de novo* transcriptome assembly (Corney, 2012), (Kim et al., 2013). Variation analysis of RNA-seq data only yields variants from exonic and UTR regions of expressed genes. Calling variants from RNA-seq data can be a by-product of gene expression studies (Quinn et al., 2013).

Using standard alignment algorithms to align RNA-seq data will fail when an attempt it made to align splice junctions with the reference sequence. *TopHat/Tophat2* is a commonly used splice-junction aware alignment algorithm in RNA-seq pipelines (H. Li & Homer, 2010), (Kim et al., 2013). TopHat2 utilises the Bowtie or Bowtie2 algorithm together with its own indel-finding algorithm which offers improvements dealing with spliced alignments. TopHat2 supports the mapping of variable-length indels (Kim et al., 2013).

4.2.3 Pre-processing

Pre-processing after read mapping may need to be performed prior to variant calling to improve the function of variant calling algorithms. Example pre-processing tasks include sorting alignments by chromosomal position, removing PCR artefacts, removing non-unique alignments and realignment of reads around small indels. *SAMtools* and *GATK* are example tools for performing

pre-processing tasks. After alignment, results can be verified with a visualisation tool e.g. IGV (Integrated Genomics Viewer) (Altmann et al., 2012).

4.2.4 Variant calling

Assuming that fragments of one or more individuals have been aligned to a reference genome (i.e. sequence assembly is excluded), variant calling can be performed. Variant calling is the identification of biological variants: Small mutations (e.g. SNPs, short indels) or structural variants (*copy-number variations (CNVs)* and genomic arrangements) in relation to a reference genome in the individuals or populations of interest (Dalca & Brudno, 2010), (Nielsen et al., 2011). Variants are called from SAM/BAM files and output to the *variant call format* (VCF), a tab delimited text format which encodes DNA polymorphism data (SNPs, indels and SVs). VCF is a widely accepted and flexible format produced by a majority of variant calling tools (Altmann et al., 2012).

Basic variant calling methods require separate analyses for individuals sampled. Filtering of aligned sequence data is performed to retain high-confidence bases by simply counting alleles at each site; the method being more or less similar to genotype calling. Simple cut-off rules are observed. *Phred scoring* (discussed in further detail in the following section) is the most common type of cut-off scoring (Nielsen et al., 2011).

Modern variant calling methods are characterised by the possibility for using multiple individuals and incorporating uncertainty (often described as a *quality score*) in a probabilistic framework coupled with prior information (Nielsen et al., 2011). Additionally, LD can offer further improvements in insufficient variant calling by allowing the imputation of missing genotypes (Altmann et al., 2012). Prior information may be sourced from databases (e.g. *dbSNP*) or carrying out variant calling on multiple individuals simultaneously (Altmann et al., 2012). Earlier methods based on counting do not provide measures of uncertainty.

SAMtools and *GATK (Genome Analysis Toolkit*) are examples of widely adopted variant calling tools capable of supporting multiple sample calling (Altmann et al., 2012). Both of these libraries are Linux-based and require installation. Neither appear to have public APIs exposed via web services (although *Galaxy* provides an online portal for GATK workflows) (Altmann et al., 2012). The *SAMtools* software package developed in C, and additionally, interfaces to SAMtools are available in other languages for example *PySam* (Python) and *RSamTools* (R). In addition to variant calling,

SAMtools contains various utilities including indexing, format conversion and an alignment viewer using SAM/BAM format (H. Li et al., 2009).

SAM is a tab-delimited format that stores information about each aligned read including the position of the reference contig, orientation of the read, quality of the alignment and potential further alignment possibilities of the read (Altmann et al., 2012). SAM files contain an optional header starting with '@', followed by the alignment section. The alignment section has 11 mandatory fields plus additional optional fields. An alignment section typically represents a linear alignment of a segment (The SAM/BAM Format Specification Working Group, 2014).

mpileup and *BCFtools* from the SAMtools package provide the functionality needed to form the variant calling pipeline. *mpileup* is capable of supporting multiple sample calling, providing data on a per-position basis and coupled with *BCFtools* providing the actual variant calling functionality. Data is produced in either .VCF or .BCF format (BCF or *binary VCF* is the compressed binary counterpart of VCF). It should be taken into account that the .VCF file produced by SAMtools does not conform to the official VCF standard ("Multisample SNP Calling," 2010). *HTSlib* is a low level C library for fast processing of high throughput sequencing file formats that provides common functionality to the *SAMtools* and *VCFtools* libraries to perform variant calling.

Genotype calling is the process of analysing genotyping data for variants. Genotype calling determines the genotype (alleles) at a specific location in an individual and often requires some prior knowledge of the variant of interest. For model organisms, high-throughput genotype chip arrays are available for common variants. As opposed to genotyping where specific locations are assessed; sequencing determines the exact sequence of a region of DNA, uncovering the exact order of the nucleotide bases (for example DNA sequencing can target a specific region of DNA, a gene, the exome or the whole-genome) (Nielsen et al., 2011). Although cost-effectiveness is a common theme associated with NGS experiments the cost-benefit should be weighed up. For example whole-genome sequencing of species with large genome size is a costly exercise compared with sequencing a targeted region of DNA (Casals et al., 2012).

4.2.5 Accuracy and errors in variant analysis

In NGS studies, a number of steps can be taken in various phases of the pipeline to improve the accuracy of NGS experiments (Nielsen et al., 2011). Errors in NGS data can arise from a multitude of factors including errors in library preparation, base-calling, read mapping and in the reference

genome sequence (Nielsen et al., 2011). All of these factors can affect accuracy of variant analysis. The resultant effects of platform errors, mapping errors and level of coverage complicate variant calling, e.g. correctly identifying zygosity likelihoods, i.e. whether an individual is homozygous or heterozygous (Dalca & Brudno, 2010).

The remainder of the chapter provides examples of the types of errors experienced in different phases of the pipeline and a revision of correction techniques.

Base-calling

Base-calling algorithms are designed to ensure accuracy by applying corrections to commonly known base-calling errors (Ledergerber & Dessimoz, 2011). In addition to the vendor-specific proprietary base-calling software, alternative third party options exist.

Uncertainty is measured during base-calling by providing a per-base quality score. Base-calling methodologies differ across vendors. Most vendors provide Phred-like quality scores (e.g. Illumina) or scores that can be easily converted into Phred scores to measure uncertainty. Some algorithms model biases (e.g. noise estimates from image analysis), others rely on reference sets to train classifiers and apply statistical learning (Ledergerber & Dessimoz, 2011). Some vendors bundle a reference genome with their software.

$$Q_{Phred} = -10\log_{10} P(error)$$

A Phred score of 30 means 1 error in 1000 bases. The most common cut-off point is a quality score of Q20, meaning an error rate of 1% (Nielsen et al., 2011).

Although the variant calling process is similar amongst all sequencing platforms, the types of base-calling errors can be attributed to the type of sequencing platform based on the platforms chemistry and mechanistic details. Understanding the characteristics of these different types of errors is beneficial to downstream analysis (Ledergerber & Dessimoz, 2011).

The most common source of error encountered in the Illumina/sequencing-by-synthesis terminal-based technologies is substitution errors (Dalca & Brudno, 2010). Errors are most likely to occur after a G-base. AT-rich and GC-rich regions are underrepresented, which can result from an amplification bias in template preparation (Benjamini & Speed, 2012).

Base calling error types

Cross-talk *is the overlapping of frequency spectra.*

Mixed clusters *are when different DNA fragments bind to a single bead.*

Phasing *occurs during sequencing where a strand which has failed to incorporate a base in a given cycle will continue to lag behind.*

Pre-phasing *occurs while sequencing, where multiple bases are synthesized in a single cell.*

Thresholding *determines if a base was incorporated or not.*

Signal decay *is the decay of signal intensity from one cycle to another.*

Source: Ledergerber & Dessimoz (2011)

Additionally, other sources of errors are typically caused by mixed clusters, phasing, pre-phasing, cross-talk (Ledergerber & Dessimoz, 2011) and desyncronization during the synthesis process (Nielsen et al., 2011). Higher error rates occur in the later machine cycles due to so-called 'cycle effects' otherwise known as *signal decay* (R. Li et al., 2009).

The most common source of errors in Roche 454/pyrosequencing technologies are high indel error rates due to thresholding errors caused by incorrect prediction of homopolymer lengths (Ledergerber & Dessimoz, 2011). Other less common forms of errors in pyrosequencing technologies include mixed clusters, phasing and pre-phasing (Ledergerber & Dessimoz, 2011). Due to fewer reads than other platforms, the Roche 454 platform has shallower coverage than other NGS technologies (Ekblom & Galindo, 2011).

Types of errors that can arise regardless of the type of sequencing platform are cross-talk, mixed clusters, thresholding and biases in chemistry leading to phasing and pre-phasing (Ledergerber & Dessimoz, 2011).

The accuracy of sequencing can be improved by increasing the coverage (e.g. > 20x coverage) but at a greater cost (Nielsen et al., 2011). Deciding on a platform is generally a trade-off between read length and sequence output (Ekblom & Galindo, 2011). Development into and the adoption of more sophisticated base-callers reduces the coverage required to reach a given accuracy.

Read mapping

Correctly mapped reads play a crucial role in variant detection with incorrectly mapped reads leading to errors in variant calling. It is imperative for read mapping algorithms to deal with sequencing errors versus potentially real differences (e.g. point mutations and indels) (Nielsen et al., 2011). Errors in read mapping typically arise as quality issues associated with low coverage

where errors bear great similarity to low-frequency SNP alleles (Ekblom & Galindo, 2011). Short read lengths inherent to NGS experiments introduce errors in the form of *misalignment* (or *miscalls*) of reads becoming apparent after the initial mapping. Types of misalignments include *multiple alignments* and *mismatches.* Multiple alignments occur due to the genomes inherent nature to contain repetitive sequences (or close to repetitive) relative to the read length that are capable of mapping to multiple positions. Mismatches are incorrectly matched reads that occur when reads containing mutations or sequencing errors result in reads being mapped to an incorrect location on the reference genome (H. Li et al., 2008). Therefore it is vital to differentiate between a mismatch which identifies a potential variant and a source of error i.e. cases where a mismatch could be a true SNP mapping (R. Li et al., 2009). Low coverage can complicate matters, especially in positions where only one diploid individual has been sequenced (Nielsen et al., 2011).

Rather than discarding ambiguous errors, it is more beneficial to retain mapped reads for evaluating the likelihood with Phred-like quality scores (H. Li et al., 2008). Quality can be further rectified with filtering actions. Contamination (due to artefacts such as primers not being successfully removed) may appear as actual differences but can be removed by filtering actions when the sequence is known. Generally misalignments can be reduced by improved experimental procedures or higher accuracy in base-calling algorithms (Nakamura et al., 2011).

Uniqueness is a measure used in determining the accuracy of alignments indicating that a read having a unique match was not subject to mismatches (R. Li et al., 2009). Choosing a sequencing methodology with *paired-end reads* (as opposed to single-end sequencing) improves the measure of uniqueness. Paired-end reads benefit from improved resolution due to distance information incorporated with sequence information (R. Li et al., 2009).

Recalibration and realignment

The pre-processing phase of the pipeline incorporates such techniques as recalibration (Figure 4.2) and realignment to improve the accuracy of variant calling.

Prior to analysis, it is advantageous to have well-calibrated, accurate quality scores to improve the function of variant calling algorithms. The ‘cycle effect’ is a phenomena where Phred-like quality scores from the base-calling stage have a tendency to deviate from the true error rate in later cycles. This example problem can be solved by recalibration (DePristo et al., 2011). Popular variant calling tools *GATK* and *SAMtools* support recalibration (Nielsen et al., 2011).

Local realignment corrects alignments around small indels. Realignment can substantially reduce errors caused by misalignments, a major source of errors in variant calling (Abecasis et al., 2010).

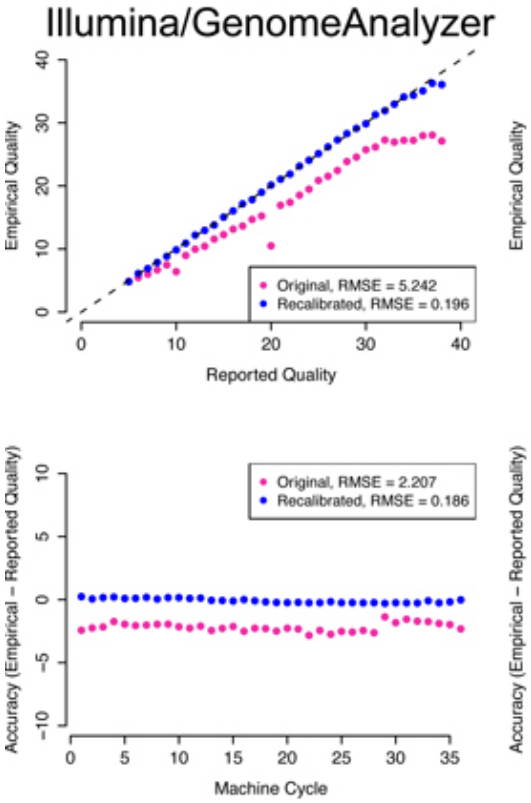


Figure 4.2 – The effects of recalibration. Raw (violet) and recalibrated (blue) base quality scores for NGS paired end read sets from 1000 Genome Project sample data set NA12878 of Illumina/GA. Top graph shows reported base quality scores compared to the empirical estimates. Bottom graph is the difference between the average reported and empirical quality score for each machine cycle, with positive and negative cycle values given for the first and second read in the pair, respectively. Root-mean-square errors (RMSE) are given for the pre- and post-recalibration curves. Reprinted by permission from Macmillan Publishers Ltd: NATURE GENETICS (DePristo et al), copyright (2011).

Variant calling

Section 4.2.4 introduced the concept of probabilistic methods for quantifying uncertainty. Generally it is required that accurate variant calling methods require sensitivity and specific statistical models (commonly Bayesian) (DePristo et al., 2011). Recently developed statistical methods improve and quantify considerable uncertainty associated with genotype/SNP calling studies (Nielsen et al., 2011).

Filtering

The filtering step performed post-variant calling reduces the number of false-positive SNP calls and SNPs based on other filter criteria. *GATK*, *SAMtools* and *VCFtools* provide filtering functionality (Altmann et al., 2012).

Traditionally SNPs are filtered based on exhibiting characteristics outside of their normal ranges, for example; low frequency alleles, bases with low base call quality, reads with low mapping quality and deviations of genotype frequencies from the Hardy-Weinberg equilibrium (HWE) in full-sib families. Filtering is often based on data that is already available and methods may be based on the sequencing protocol in use and upstream analysis methods (Nielsen et al., 2011)

Hardy-Weinberg equilibrium (HWE) *is a crucial concept in population genetics that suggests that probable allele frequencies in populations from generation to generation remain constant if factors (selection, genetic drift, gene flow, and mutation) do not effect allele frequency (Levin, 2009).*

Full-sib *is a sibling with both parents in common.*

Read depth (coverage) *is the average number of reads a region has been sequenced (Harbers & Kahl, 2012).*

An example application of filtering is detection of true heterozygote alleles. True heterozygote alleles are a challenge to detect from sequencing errors at low read depth. This can be compensated for by applying filtering (R. Li et al., 2009). Applying 'hard' filtering (with strict cut-off rules) can be a trade-off that can be counter-balanced with quality score recalibration (DePristo et al., 2011).

4.2.6 Analysing SNP data

SNP analysis forms the final stage of the SNP/Genotype calling pipeline that attempts to make sense of the data from analysis-ready called variants (Altmann et al., 2012).

When performing variant analysis, SNPs are commonly classified according to their location in the genome. The VCF file format stores information regarding the chromosomal position, reference base, alternative bases, quality score, read depth and additionally may include metadata; additional information described in arbitrary and reserved key-value pairs, and genotyping information. SNP annotation tools can be utilized further to make sense from the vast amounts of variant data generated from sequencing experiments (Danecek et al., 2011). For further information on the VCF specification, refer to section 5.3.2. An exhaustive list of downstream VCF analysis techniques will not be discussed, but it is useful to be aware that the VCF format is flexible and supported in many different software tools.

VCFtools is an open-source software package for the manipulation, parsing and analysis of SNP data from VCF/BCF formats. VCFtools provides scalable support for multiple samples (Danecek et al., 2011).

VCFtools provides a Perl API for file manipulation purposes and a C++ binary for providing deep analysis of SNPs including estimation of allele frequencies, levels of LD and quality control metrics (Danecek et al., 2011). The VCFtools Perl modules require the *bgzip* and *tabix* utilities packaged with *SAMtools* which provide compression and storage of VCF files and indexing and fast retrieval of data based on genomic position (VCFtools, 2014). The benefits of indexing provides the ability to read specific genomic regions of interest, as most biological data formats contain chromosomal positions; when analysing large datasets without the need to load an entire input file into memory (H Li, 2011).

Discussed below is a review of existing tools which allow searching by location and/or other SNP properties.

SnpSift is a set of tools written in Java to locate variants of interest based on annotated data. SnpSift requires that VCF is annotated with *SnpEff (Genetic variant annotation and effect prediction toolbox*) belonging to the same suite of tools. SnpSift allows the filtering of SNPs based on common criteria (e.g. quality) and metadata (e.g. VCF *INFO* fields). Individual genotype data is exposed in a *GEN[]* array object. The use of wildcard expressions provide the ability to search multiple genotype fields (Cingolani et al., 2012).

SNPPer is a web-based analysis tool which allows the user to filter SNPs by position, name or other properties (Riva & Kohane, 2004). The limitation is that the tool is only applicable to the human genome and evidence suggests that the tool is not actively developed.

PupaSuite (an amalgamation of existing tools *PupaSNP* and *PupasView*) is a web-based analysis tool for locating SNPs with potential phenotypic effect linked with *SNPEffect* database predictions. The tool allows the user to perform SNP prioritization based on functional properties, type, validation status and population. Additionally, information on LD parameters and haplotype blocks is available. PupaSuite aims to aid in the design of large-scale genotyping projects and is applicable to human, mouse and rat genomes (Conde et al., 2006).

The 1000 Genome Project web site contains a number of online web-based tools for analysing VCF data. The tools are specific to the project and require an associated *.panel* file to filter results by release and population. Selecting subsections of VCF based on genomic coordinates with the Data Slicer tool requires a corresponding index file (Clarke et al., 2012).

Evidently there are existing databases and tools that were capable of searching by chromosomal location or filtering with population data, but there was no evidence sourced of any existing tools that would integrate non-model organism variant data with population metadata that would meet the aims of the thesis.

5 Technical Solution

This chapter discusses the methods used for solving the project goals. Firstly, the project goals are defined in detail. Secondly, the input data sources (VCF and metadata files) are further described in order to provide the background on the requirements for the data model.

The crux of the chapter discusses the implementation details. The solution architecture tiers are discussed: The web site (presentation layer), application logic (business rules) and database (storage) tiers. Furthermore, the process for arriving at the chosen technology stack and comparison of different database paradigms is discussed as the database and storage tier is the main tenet in designing the solution architecture. Performance considerations encountered during development and general best practices are discussed as performance considerations are essential to query response time. Finally the functional requirements are discussed, explaining how the application satisfies the goals of the thesis.

5.1 Aim

The aim of the project is to develop a solution for the data management and tool for searching heterogeneous data sets for further analyses. The data for the thesis project originates from the RNA-seq study of the Glanville fritillary (Kvist, 2014)

The presentation layer is developed for the web to provide ease of access without the requirement to install specialised software. The primary reasons for selecting a database platform for storage is structuring data, handling large datasets, defining relationships, scalability, accessibility, queryable and otherwise provide storage for the analysis of datasets that otherwise would not be publicly available. The intended audience is researchers and lab staff.

The solution aims to allow researchers to:

- Study individual variation between and within populations.
- Study variation in the level of habitat patches in the Åland Islands.
- Verify locations of variation in scaffolds.
- Select interesting SNPs from candidate regions.
- Include fields from VCF metadata (e.g. INFO fields).
- Aid in the design of genotyping and resequencing experiments.
- Search multiple input files.

- Export search results to .CSV format.

The solution should provide the ability to search for variants from several input files without requiring expert knowledge to use the system, be well-documented to allow future development and provide an administrative interface for the uploading and management of data files. The requirements were platform agnostic, allowing flexibility in selecting a solution stack.

5.2 Functionality

This section discusses the solution implementation from a functional and usability perspective on how the researcher would interact with the web site to query variation. The technical implementation details are discussed in section 5.4. Application screen shots featuring images of the most common views are located in Appendix A.

5.2.1 Site overview

The web site landing page provides links to common functionality including search, data sources management, help content and the site overview.

All of the functionality for querying variants is built into the search page. The primary function of the search page is to display aggregated data from VCF and metadata filtered via population information entered by the user. The search page is divided into three sections: *Data Sources, Filters* and *Advanced Filters*. The sections are displayed with a concertina effect (i.e. the sections are expandable by clicking on the section headings or arrow icons). Only one section is displayed at a time, with the data sources section expanded by default as this will be the most frequently used section of the page.

5.2.2 Data sources section

The data sources section (Figure A.1) displays a table listing showing available data files. In the first release, the table is informational only. In future releases, the user will be able to select data sources from this view.

5.2.3 Filters section

The filters section (Figure A.2) provides the most common filters (Table 5.1). Users can filter search results by selected VCF fields (*Range Begin, Range End, Scaffold* and *Quality Score*) and the more commonly used Progeny metadata fields (*Gender* and *Population*).

FIELD NAME	DATA TYPE	DATA SOURCE	CORRESPONDING FIELD FROM DATA SOURCE	DESCRIPTION
RANGE BEGIN	Integer	VCF	POS	The starting position
RANGE END	Integer	VCF	POS	The ending position
SCAFFOLD	String	VCF	CHROM	Multi-select populated from database
GENDER	String	Progeny	Gender	Pre-populated multi-select (M, F)
QUALITY (PHRED) SCORE	Double	VCF	QUAL	Default: 20
POPULATION	String	Progeny	Population	Multi-select populated from database. Look-up table provides population full names from abbreviations.

Table 5.1 – List of fields displayed in the filters section of the search view.

5.2.4 Advanced filters

The advanced filters section (Figure A.3) allows the user to filter by additional population parameters including *Patch, Patch Commune* and *Patch Network* (Table 5.2).

FIELD NAME	DATA TYPE	DATA SOURCE	CORRESPONDING FIELD FROM DATA SOURCE	DESCRIPTION
PATCH	String	Progeny	Patch	Text-box with autocomplete
NETWORK	Integer	Progeny	Patch.Network	Text-box with autocomplete
COMMUNE	String	Progeny	Patch.Commune	Text-box with autocomplete

Table 5.2 – List of fields displayed in the advanced filters section of the search view.

5.2.5 Search results

The search results view (Figure A.4) displays the DataTables data grid populated with filtered VCF information stored in the database (Table 5.3).

The search results table is not displayed until after the search parameters are defined and the user selects the *Search* button to search the database. The user can resubmit the search simply by defining new parameters and reselecting *Search*. The *Clear* button clears the form completely.

The available columns in the table include scaffold ID (CHROM), variant position (POS), reference alleles (REF), alternate alleles (ALT) and quality score (QUAL) from VCF mandatory columns and allele frequency for alternative allele (AF) and depth (DP) originally parsed from VCF INFO metadata. GT is a calculated field that displays the number of individuals with different genotypes (Ref/Ref, Ref/Alt, Alt/Alt) where genotype quality (GQ) is assumed to be >=20 and depth (DP) > 0 from genotype fields. Section 5.3.2 discusses the VCF specification in further detail.

The search results are displayed in a tabular format. A drop-down list at the top of the table allows the user to specify the number of results to display in the table (10, 25, 50 or 100). By default the search results are filtered by quality score (QUAL) >= 20 which can be overridden by the user in the search parameters.

FIELD NAME	DATA TYPE	DATA SOURCE	CORRESPONDING FIELD FROM DATA SOURCE	DESCRIPTION
VARIANT ID	Int	*PK*	-	Displayed in the drill-down view. Auto-generated key value to uniquely identify records from VCF.
SCAFFOLD	String	VCF	CHROM	Scaffold Id
POS	Integer	VCF	POS	Variant Position
REF	String	VCF	REF	Reference allele(s)
ALT	String	VCF	ALT	Alternate allele(s)
AF	Integer	VCF	INFO / AF	Allele frequency for alternate allele
DP	Integer	VCF	INFO / DP	Depth
QUAL	Double	VCF	QUAL	Quality (Phred-scaled) score
GT	String	VCF	Genotype / GT	Calculated from the number of individuals with different genotypes (Ref/Ref, Ref/Alt, Alt/Alt) where genotype quality (GQ) is assumed to be >=20

Table 5.3 – List of fields displayed in the data grid in the Search View.

5.2.6 Export

The *Export* button only appears after the search results data grid is populated. Selecting *Export* exports all filtered results from the data grid and not just the current page, to CSV. During export a process indicator appears until the user is prompted to download the file.

5.2.7 Help

The user can gain context-sensitive help by mousing over labels or by selecting the help icon in the header bar.

5.3 Data sources

This section describes the data sources (Table B.1) used to populate the database. Section 5.3.1 describes the experimental procedures for producing the variation data. Section 5.3.2 describes the VCF specification and section 5.3.3 introduces the Progeny LIMS system.

5.3.1 Background

The original intention was to include VCF data produced from RNA-seq and resequencing experiments. To reiterate, RNA-seq and resequencing are NGS methods that can be used in variation analysis. The end result is variant calling which is essentially the same as DNA-seq, resulting in a VCF file with called variants except the specific algorithms and tools for variant calling differ.

The data was prepared for a gene expression and variation analysis study of the Glanville fritillary (Kvist, 2014). The butterflies used in this study originated from the large metapopulation in the Åland Islands (AL) in Finland (Ilkka a Hanski, 2011) and from a small, completely isolated population on the small island of Pikku Tytärsaari (PT) in the middle of the Gulf of Finland (Mattila et al., 2012).

Two RNA-seq libraries with insert sizes of 450 bp and 650 bp were sequenced for each individual. Sequencing of the RNA-seq data was carried out with the Illumina HiSeq 2000 platform using the paired-end mode with 2 x 101 bp read length. Only the 3' end of each transcript was sequenced. Read pairs were mapped against genomic scaffolds (Ahola et al., 2014) using Tophat2 (Kim et al., 2013). Variants were called using SAMtools (H. Li et al., 2009).

5.3.2 Variant Call Format

The VCF file format (Figure B.1) is a tab-delimited text format that encodes DNA polymorphism data, typically stored in a compressed form. To recap on how VCF fits in with the variant calling pipeline, refer to section 4.2.6.

The file contains a header, meta-information fields and data presented line-by-line representing a position in the genome. Additionally, the format may contain genotype fields representing individual samples at each position. Meta-information appears before the header as key-value pairs prefixed with hashes '##'.

The header line names the 8 mandatory columns (CHROM, POS, ID, REF, ALT, QUAL, FILTER, and INFO). A period ('.') denotes missing values. The INFO field contains additional information and is broken down into key-value pairs delimited by semi-colons. Key values can be arbitrary or reserved words. The INFO fields required for the project are the AF and DP values (Table 5.3). In the case there are multiple metadata values in the INFO field i.e. the field is appended with an integer e.g. AF*1,* the first instance of the field is taken.

If genotyping data is present, firstly the FORMAT column must be present indicating the available fields. Similarly to the INFO fields, genotype fields contain delimited key value pairs. Genotype columns are listed as an arbitrary number of sample IDs. Data specified in the FORMAT column must be consistent across all samples (Samtools, 2013).

The supplied data file for the study contains genotyping information (Kvist, 2014). The GT (genotype), GQ (genotype quality) and DP (depth) are used to describe and/or filter variants from the database. The column header contains the sample ID (composed of a BAM file name) relating to the individual sample in the genotyping data. The sample ID correlates genotyping data with metadata; such as sex, population and habitat patch information downloaded from Progeny database based on the *Individual Name*.

5.3.3 Progeny

Progeny Lab is a proprietary Laboratory Information Management System (LIMS) for the centralised management of genetic data. Progeny Lab natively supports genotype data from major vendors (including ABI and Illumina). Progeny Lab provides analysis tools for SNPs and STRs (short-tandem repeats)

The prepared metadata for the project has specific columns (*Individual Name*, *Gender*, *Patch*, *Patch.Commune, Patch.Network,* and *Population*) selected from Progeny Lab and exported as text (Figure B.2). There is not a specific file format nor was it possible to integrate directly with Progeny.

5.4 Technology stack

The database model was built on relational database platform SQL Server 2014. The web interface is developed in ASP.NET MVC 5.0. In order to populate the database with data, ETL

ETL (Extract, Transform, Load) *is a process where data is extracted from various data sources, transformed into an appropriate structure for querying and loaded into the target database (Horváth, 2013).*

(*Extract, Transform, Load*) operations are performed using SSIS (SQL Server Integration Services). The ETL operations are discussed in further detail in section 5.4.1.

The motivation for selecting SQL Server 2014 is discussed in section 5.4.4 where differing paradigms, relational versus NoSQL database are compared. ASP.NET MVC was selected as a modern, light-weight, presentation framework with support for testability.

Additional libraries required to develop the working solution include:

- Entity Framework 6.1
- LinqToCsv 1.5
- DataTables 1.10.0
- DataTables.Mvc 1.0 (forked)

The library functionality will be discussed in further detail throughout the chapter.

The presentation layer and application logic was developed in Microsoft Visual Studio 2013. A free version is also available (*Visual Studio Express*). The SSIS packages for ETL were developed in SQL Server Data Tools for Visual Studio 2013, an add-on for Visual Studio.

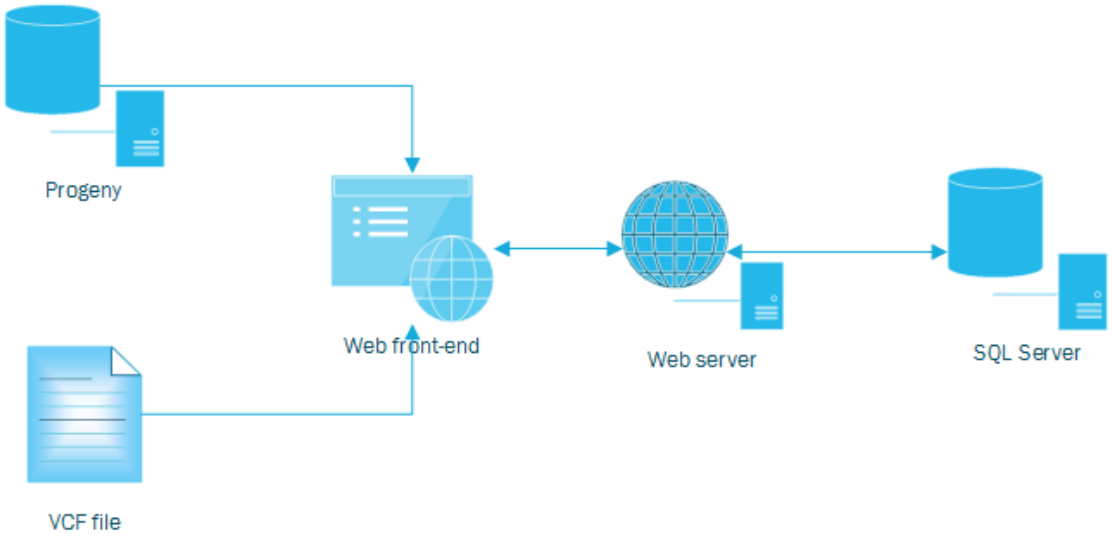


Figure 5.1 - Logical topology diagram of the planned architecture. Progeny and VCF data is entered via the web front-end which is then delegated to SQL Server to be parsed.

From here on in, the term *solution* is used to refer to the Visual Studio solution that acts as a container for software development artefacts. *NuGet* is a package manager for Visual Studio (Outercurve Foundation, 2014) that eases the process of installing and updating third party

libraries in the solution and it will be stated throughout this chapter when NuGet is used to install packages.

5.4.1 ETL

ETL is a process where data is extracted from various data sources, transformed into an appropriate structure for querying and loaded into the target database (Horváth, 2013).

This section introduces the Microsoft SQL Server Component *SSIS* and describes the SSIS package design which essentially controls the execution of ETL operations.

Introduction to SSIS

SSIS (SQL Server Integration Services) is a component of SQL Server providing powerful data integration and migration operations with the ability to build complex workflows between heterogeneous data sources and destination. In this case, SSIS was selected as a tool to perform ETL operations on data from VCF and Progeny data sources to a SQL Server database. SSIS was selected due to the fact it is a native component of the SQL Server stack, has good performance and due to familiarity with the product as a commercial tool. SSIS contains built-in tasks that eliminate the need for vast amounts of custom code. SSIS supports parallel execution with operations running in memory space, eliminating the need for a staging area to transform data. Additionally, SSIS can be ran independently of a SQL Server instance (Microsoft, 2014i).

SSIS package design

An SSIS package is a container for *control flow elements* and *data flow elements*. Control flow elements control the overall flow of the package. Dataflow elements describe fine grained operations on data that belong to a parent control flow element. In its simplest form, a SSIS package contains a data source and a SQL or data flow task and a destination (Microsoft, 2014j).

The developed SSIS solution consists of two packages: *VCFExport.dtsx* for exporting VCF data and *ProgenyExport.dtsx* for exporting Progeny data. In the solution, the destination for data flow tasks are configured to target a previously created variation database (described in section 5.4.4).

The packages are developed using the visual designer in SQL Server Data Tools for Visual Studio 2013 which supports drag-and-drop components. Appendix C illustrates the designer views as this serves as a method to communicate the data flow to the reader.

Essentially whenever the database needs to be updated, the flat files sources need to be updated to point to the new data source. Packages should be designed in such a way that they are executed *independently* of the designer and passed input configuration parameters.

VCFExport package workflow

The *VCFExport.dtsx* package controls the workflow for extracting all relevant data from VCF and importing the data into the SQL server database. The workflow is summarised below and illustrated in Appendix Figure C.1:

- *Insert File Task* is an SQL task that issues an SQL INSERT statement to populate the DataFile table by specifying a unique ID and file name for the originating data source. In reality, the file name should be passed as a parameter from the data source but for the proof of concept this will suffice.
- *Populate Variant table* is a data flow task that populates the *Variant* table.
- *Populate Genotypes table is* a data flow task that populates the *Genotypes* table.
- *Calculate Total GT* is an SQL task that performs post-processing operations to calculate the total GT (genotype) from the GT genotyping column in VCF.

Populate Variant table data flow task

The *Populate Variant table* data flow task performs ETL operations between the VCF flat file and the SQL database targeting the *Variant* table.

The workflow is summarised below and illustrated in Appendix Figure C.2:

- *VCF Flat File Source* describes which columns should be selected from the VCF file. The connection manager is configured to understand the encoding, delimiter and starting position for parsing. The task is straight-forward to configure and requires no additional scripting.
- *Assign file IDs and parse INFO Script Component* is a script component that parses incoming rows. The script component written in C#, parses the AF and DP fields from the VCF INFO metadata field with regular expressions.
- *Convert strings to Unicode* is a data conversion task that ensures that input columns are in the correct format for the database. This extra step is taken because the VCF file source was non-Unicode.

- *Variant Table Destination* maps input columns to destination columns and loads data into the destination Variant table.

Populate Genotypes table data flow task

The *Populate Genotypes table* data flow task performs ETL operations between the VCF flat file and the SQL database targeting the *Genotype* table.

The workflow is summarised below and illustrated in Appendix Figure C.3. The workflow is quite similar to the *Populate Variant table* task described above except:

- *Unpivot individuals to rows* converts columns to rows (in this case, the genotype fields from VCF are transposed from columns to rows). Transposition is required in order to create the relational database model.
- After unpivoting, rows can be processed with *a data transformation task*. The script component uses string operations and regular expressions to parse the GQ, DP and GT metadata from genotype fields. The column heading from VCF (containing the BAM file name) is truncated to match the individual name in Progeny to later serve as a relationship between Progeny and VCF data.

Calculate Total GT execute SQL task

The *Calculate Total GT* SQL task (Figure C.4) makes use of a temporary table to store calculated GT values. The temporary table is populated by using an SQL *INSERT* statement combined with a *SELECT* statement (retrieves records) that calls a *user-defined function* (UDF) (Microsoft, 2014p) to calculate the GT (Figure C.5). The result is returned as an *nvarchar* data type.

The UDF counts the number of individuals with different genotypes (*Ref/Ref, Ref/Alt, Alt/Alt*) by counting the combinations of allele values based on the metadata criteria that DP (depth) is > 0 and GQ (conditional genotype quality) is >= 20.

After the temporary table is populated, an UPDATE statement is called on the *Genotype* table, setting the *CalculatedGT* value to the previously calculated value from the temporary table. The temporary table is *dropped* (removed) after the operation is complete.

The example syntax for the execute SQL task and UDF is shown in Appendix C.

Processing the Calculated GT in SQL and storing the values in a table provides the advantage of not having to perform aggregations during runtime thus prevents potential performance issues.

ProgenyExport package workflow

The *ProgenyExport.dtsx* package controls the workflow for extracting all relevant data from the Progeny flat file source and importing the data into the SQL server database. The Progeny workflow is very similar in practice to the *Populate Variant table data flow task* and does not require any complex transformations.

5.4.2 Website

ASP.NET MVC

The presentation layer is developed for the web to provide ease of access without the requirement to install specialised software. This section describes the various technologies used to implement the web interface. The web interface is developed in ASP.NET MVC 5.0. ASP.NET MVC was selected as the web framework due to familiarity with the product as a widely-adopted framework in industry.

ASP.NET is Microsoft's offering of a web based development framework based on the .NET framework. The .NET framework is a software framework developed by Microsoft that primarily runs on Windows (Microsoft, 2014e). *MVC* (model view controller) is a common development paradigm for the isolation of concerns between the UI (user interface) layer and the application logic. ASP.NET MVC does not have a concept of a *page* existing on a physical disk. Instead the component responsible for displaying the UI is known as a *view*. Views contain elements of the *model* to the user. To simplify, models represent the data whereas controllers perform the routing of requests.

The latest version of ASP.NET MVC supports other modern web best practices including *Modernizr*, a library for supporting older browsers and *Bootstrap*, a front-end framework for front-end design supporting responsive layouts and theming. ASP.NET MVC 5 requires to be hosted on Windows Server (the pilot is hosted on Windows Server 2012 R2).

ASP.NET MVC uses the *Razor* view engine, a set of conventions used for embedding dynamic content in HTML. A *Helper* is a feature of the Razor view engine for encapsulating the rendering of HTML in a reusable template to avoid repetition. ASP.NET MVC is bundled with a set amount of

default helpers. Custom Helpers can be used to build custom controls by encapsulating rendered HTML and server-side logic. The advantage of using a Custom Helper is that if it conforms to the MVC architecture, it should provide cleaner HTML, separation of concerns and better testability (Freeman, 2013). The concept of a Helper is important here because third party libraries can be provided as libraries of Helpers. For example, the grid control itself is loaded onto the main (*Index.cshtml*) view with a Helper, where the *Action* helper invokes a child action from the *controller*:

```
@Html.Action("DataTable")
```

The corresponding controller method, simply returns a partial view containing the HTML mark-up for the table where a partial view is like a regular view but it can be used to encapsulate reusable content or embed in parent views:

```
public PartialViewResult DataTable()
{
       return PartialView("_DataTable");
}
```

Grid Control

Three data grid controls (for displaying search results) were evaluated. *Grid.Mvc* is an open source Helper grid control implementation for ASP.NET MVC. (Bukharin, 2014). *DevExpress ASP.NET Grid View extension* is a proprietary Helper grid control forming part of *DevExpress development tools software* (DevExpress Inc, 2014). *DataTables* is an open-source client-side plugin developed in jQuery, a JavaScript library for the easy manipulation of HTML (SpryMedia, 2014). All the grids exhibit functionality atypical of grid controls (e.g. pagination, sorting, filtering and theming). DataTables and Grid.Mvc is available from NuGet. DevExpress is available as a free trial from the DevExpress web site.

The goal of trailing the different grid controls was to select a performant grid control that fit the needs of the project. Out of the three controls tested, DataTables was the most performant, which is a purely client-side implementation.

Although Helper implementations exist for DataTables, in this case DataTables was configured according to the online manual for client-side inclusion and customized to suit the project requirements for the reason being that during development the control performed better when

placed directly into HTML. The advantage of client-side controls over server-side controls it that they involve less trips to the server. The implementation embraces *AJAX* calls which prevents the need for the whole page to be refreshed to update partial content.

DataTables was configured to use the *server-side* data source option which offloads processing to the controller where processing large datasets can be delegated to the database server rather than the client (SpryMedia, 2014).

5.4.3 Application logic

The application code-behind is developed in C# targeting the .NET framework 4.5.1. C# is a strongly-typed object oriented language developed by Microsoft (Microsoft, 2014e).

Entity Framework

Database logic is implemented in Entity Framework version 6.1. Entity Framework is ORM (object relational mapper) software from Microsoft which provides a layer of abstraction between the database and application logic, allowing developers to work with domain-specific objects (Fancey, 2010).

Linq to Entities is a provider enabling Linq syntax for writing strongly-typed queries against the Entity Framework object model. Linq (*Language Integrated Query Language*) greatly simplifies writing queries against disparate data sources (SQL, XML, web services etc.) by providing a common query language construct in C#. Linq to Entities is the Entity Framework flavour of Linq used for querying databases. Linq to Entities removes the requirement of writing much of the database 'wiring' code and builds the SQL query syntax on behalf of the developer. This prevents the need to issue pure SQL commands from the application, protecting applications against potential SQL injection attacks (Microsoft, 2014f).

Linq to Entities is not without fault where on occasion the Linq queries can be translated to poorly performing SQL. In this scenario some understanding of SQL is useful and the developer should profile the SQL output in order to tweak the Linq query.

The application logic uses a simple version of a *repository* pattern. The controller does not interact with the data model directly, it creates an instance of the service which handles the method calls to interact with and fetch data from the data model. The pattern decouples the logic that services

user interface code and database logic. The repository class is defined by an *Interface* which can ease testing and maintenance further down the track.

The *System.Linq* namespace contains two classes including Linq query operators: *Enumerable* and *Queryable* that operate on objects implementing *IEnumerable<T>* and *IQueryable<T>* where *T* is a given type. Both support *deferred* execution meaning that a query can be previously defined but results are not realised until the data is requested. The difference between the two objects is that *IQueryable* is better suited for querying remote resources and is required for Linq to Entities in order to issue queries directly against the database. *IEnumerable<T>* is executed on *Linq to Objects* and is better suited to in-memory objects (Microsoft, 2014k).

Due to deferred execution, complex Linq queries can be built that only materialize when finally needed. For example, *where* clauses are additive in the sense that they can be built based on multiple input parameters. Queries can also be sorted and shaped before being returned to the caller.

```
query = query.Where(v => v.Pos >= requestModel.RangeBegin);

query = from v in query
        where v.Pos >= requestModel.RangeBegin
        select v;
```

Figure 5.2 – Method versus query syntax. The first line of code illustrates *method* syntax. The second line of code illustrates *query* syntax. Each example is semantically similar.

Linq to Entities supports two formats for writing queries that are semantically identical: *Method* syntax and *query* syntax (Figure 5.2). Query syntax is declarative which may be considered easier to read. Query syntax is translated to method calls, otherwise known as standard query operators at compile time (Microsoft, 2014f).

When the data needs to be retrieved e.g. show the first 10 rows in a data grid partitioning methods *Skip(n)* and *Take(n)* specify the starting position and the number of rows to display. Aside from displaying only the data that is required, only retrieving required records reduces the load on the database. The current query logic builds conditions based on AND and not null. This could be extended to include OR and other logical operators in the future shall the need arise.

Solution structure and code snippets of application logic are covered in Appendix F. Class diagrams are provided in Appendix E.

LinqToCsv

LinqToCsv is an open source library for easily reading and writing to CSV file available from NuGet.

The *Write* method accepts IEnumerable<T> and IQueryable<T> where *T* is the container class modelling the object to write. Control of exported fields/properties is influenced by decorating members with the *CsvColumn* attribute in the model class (Perdeck, 2014).

DataTables.Mvc

DataTables.Mvc is a library for binding DataTables with MVC controllers (Matos, 2014). Using a library for binding prevents the need for interrogating HTTP request parameters directly. This keeps development techniques within the ASP.NET MVC ecosystem and aids testability.

5.4.4 Database

In this section differing database paradigms, relational and NoSQL databases are introduced and the motivations for selecting relational database as the storage platform is justified. The section concludes with a detailed discussion on the database design.

Relational database management systems

A relational database management system (RDBMS) is a type of database management system based on the relational model proposed by E.F. Codd (Codd, 1970). Oracle is the current market leader RDBMS with proprietary solutions Microsoft, IBM and open source MySQL retaining significant market shares. Microsoft's SQL Server targets only the Windows operating system market although connectivity drivers exist for other operating systems (Masters Emison, 2014). SQL (Structured Query Language) is the supported query language for a majority of relational databases platforms.

The RDBMS data structure is formally described into what is referred to as the database *schema*. The schema provides a distinct namespace to facilitate the separation, management and ownership of database objects (Redman, 2008). This representation of a database schema relates to Microsoft SQL Server (the selected implementation platform). Other RDBMS vendors' schema definitions may vary.

Data is structured into *tables* comprised of *columns* and *rows*. A row, also known as a *record* consists of a number of fields. *Normalisation* is the process of organizing data into two or more tables and defining the relationships between them to minimize redundancy. *Relationships* link

related data stored in one or more tables via the definition of *primary* and *foreign keys*. Primary and foreign keys are a type of constraint. Constraints are used to specify business rules and enforce relationships between tables. *Cardinality* describes the uniqueness of values between tables, for example relationships between tables can be described as *one-to-many* and *many-to-many*. A *join* is a clause in SQL that combines records from two or more tables.

Other types of objects included in a schema may include views, indexes, triggers, programmability (aggregate operations, functions and stored procedures), queues, statistics and synonyms (Redman, 2008).

Database operations are described as *ACID (*Atomicity, Consistency, Isolation and Durability*),* a set of properties that describe the transaction model in RDBMS ensuring reliability in processing transactions (Microsoft, 2014l).

The first commercial RDBMS was released by Oracle in 1979 (Oracle, 2007). It is well known that RDBMS requires a schema to be created before data can be added. Limitations imposed by the definition of the schema means that RDBMS does not lend itself well to dynamic data structures. Additionally, the modelling of relationships is not seen as a natural fit for all data sets. The dynamic nature of NoSQL is illustrated in the *MongoDB* section of this chapter.

NoSQL

The modern NoSQL movement began in 2009 to address the shortcomings of relational database. The key characteristics of NoSQL databases are schemaless (supporting both structured and non-structured data), non-relational, scalable, distributed, open source and non-ACID. NoSQL comes in many flavours, Nosql-database.org (2014) lists 150 databases as of writing.

NoSQL databases can be loosely grouped into the following categories: Column-oriented, document-oriented, key-value and graph (Nosql-databases.org, 2014).

MongoDB

MongoDB (version 2.4.8) was selected for evaluation due to the fact it is widely adopted and considered to be the leading NoSQL database, with a strong development community and documentation.

MongoDB is a cross-platform, scalable *document-oriented* NoSQL database with driver support in a number of programming languages. In order to query MongoDB with ASP.NET MVC, the *C# and*

.NET MongoDB Driver needs to be installed in the solution (MongoDB, 2014b). The driver is available from NuGet.

In document-oriented databases, a *document* represents the notion of a *record* in RDBMS. A document is encapsulated and encoded in a particular format. MongoDB documents are represented as *BSON* (binary JSON) which is a compressed form of JSON (JavaScript Object Notation) (MongoDB, 2014d). JSON is an open-standard, human readable transport format offering a lighter-weight alternative to XML. *Collections* are comprised of many *documents*. Collections can be considered analogous to tables in RDBMS. Documents support data up to 16MB. Therefore VCF file contents cannot be stored in a single document as file sizes are often in excess of 1 GB. An alternative option would be to parse VCF files, storing each row in VCF as a separate record.

MongoDB lends itself well to working with dynamic data sets. Strongly-typed entity classes can be mapped to Mongo documents (Figure 5.3). The *BsonExtraElements* attribute collects all unmapped fields from the entity class and stores them in a collection (MongoDB, 2014b). In the following examples, it is assumed that a document containing rows from VCF exists. This is to provide an example of how to work with the C# driver in MongoDB and not part of the end implementation.

```
public class MappedRead : MongoEntity
{
    public MappedRead()
    {
    }

    [BsonSerializer(typeof(StringOrInt32Serializer))]
    [BsonElement("scaffold")]
    public string Scaffold { get; set; }


    [BsonElement("pos")]
    public int Pos { get; set; }

    [BsonExtraElements]
    public IDictionary<string, object> CatchAll { get; set; }
}
```

Figure 5.3 – Simplified example of a Mongo entity class. Simplified for the sake of brevity, the primary fields from VCF decorated with the *BsonElement* attribute are mapped directly to the corresponding properties whilst all additional columns (e.g. genotype information) get caught in the *CatchAll* collection for further processing, hence there would be no need to understand exactly how many individuals are represented in the genotype data.

In a similar fashion to Entity Framework, MongoDB supports Linq queries (Figures 5.4, 5.5) (MongoDB, 2014b).

```
public IEnumerable<MappedRead> GetVariationDetails(int limit, int skip)
{
    var mappedReadsCursor = this.MongoConnectionHandler.MongoCollection.FindAll()
        .SetLimit(limit)
        .SetSkip(skip)
        .SetFields(Fields<MappedRead>.Include(g => g.Id, g.Pos, g =>
g.Scaffold));
    return mappedReadsCursor;
}
```

Figure 5.4 – Example method to query MongoDB to retrieve specific fields. Based on the entity class illustrated in Figure 5.3, the *SetFields* extension method illustrates how specific fields are returned to the collection.

```
public IEnumerable<MappedRead> GetVariationDetails(int limit, int skip)
{
    IEnumerable<MappedRead> mappedReadsCursor = (from m in
    this.MongoConnectionHandler.MongoCollection.AsQueryable<MappedRead>()
       select new MappedRead
       {
              MajorMinorAlleles = (from c in m.CatchAll where (int)c.Value != 0
              && c.Key.StartsWith("minor") select new { Key = c.Key, Value =
              (int)c.Value }).ToDictionary(t => t.Key, s => s.Value), Gene =
              m.Gene
       });
    return mappedReadsCursor;
}
```

Figure 5.5 – Example method to query MongoDB to retrieve dynamic fields. Based on the entity class illustrated in Figure 5.3, this example uses *select new* to create an anonymous type instance that selects objects from the *CatchAll* object that meet a certain condition. In this case, string matching is performed based on a given key ("minor") and if a match is found, the key/value pair is stored to a new object. This example is somewhat arbitrary and the conditions could be substituted to match other fields caught in the *CatchAll* object.

Despite the advantages of using anonymous types to read unstructured data from MongoDB, the issue still remains that extensive data parsing during runtime incurs a performance hit.

Files larger than 16 MB can be stored in *GridFS,* a storage technique in MongoDB where data is stored in 255 KB chunks over a collection of documents (MongoDB, 2014e). Uploading documents via the object model to a GridFS collection was quite straight-forward. The ability to define meta-data to uploaded files provides ease in categorising and managing files. The *OpenText()* method (from the *MongoGridFSFileInfo.OpenText* namespace) inherits from the C# *StreamReader* class (MongoDB Inc, 2014). The drawback to storing large data files in GridFS with the C# driver, is that

the inherited *StreamReader* class (From the *System.IO.StreamReader* namespace) is limited in the respect that the entire file has to be read line-by-line in order to retrieve something as simple as a count of records based on a given criteria which is not performant for large files (10 – 15 seconds on average to load ~500,000 rows[1]).

The *StreamReader* object with the while loop construct is the most performant way in C#/.NET framework to read file contents as *StreamReader* is already optimized with an internal buffer. During benchmarking file reading methods, no apparent performance benefit was gained by specifying buffered reader parameters explicitly (Lozinski, 2013). On the given size of the dataset and system resources, parallelisation of processing had no apparent effect on performance of a 789MB text file. A possible work-around would be to save the row count in a session variable for subsequent requests sent to the browser, but this would still cause the first load when each query is built to be slow.

MongoDB supports the indexing of the *_id* (id of the parent document) and *n* (sequence number) fields of the GridFS "chunks" (the constituent file parts) collection, but not the file content itself (MongoDB, 2014e). In addition to the delay imposed by reading an individual file line-by-line, scalability of searching multiple files would pose a challenge. Reading the entire data file into memory with the .NET framework risks exhausting system resources and throwing an out of memory exception. Therefore, the current implementation of GridFS does not scale to performing data analysis of large data files. It is possible to employ the use of other disk-based storage technologies that support indexing or using third party tools to integrate with MongoDB to provide indexing of full-text searching but this was not investigated as it is out of the scope of the project.

Performing ETL with the C# driver was exceptionally slow when compared with ETL using SSIS (> 1 hour to parse ~500,000 records). Proper benchmarking was not conducted as the performance was visibly slower with little point remaining to continue with this approach.

MongoDB does not support the concept of *joins* between collections. It is expected data is stored in a denormalised manner with related data stored within documents unless data in documents is structured in such a way to create manual references to relate documents (MongoDB, 2014c).

[1] This metric was tested during development, refer to Appendix D for system resources.

Denormalisation increases redundancy meaning that data structures may not be the most efficient in terms of storage on disk.

All MongoDB distributions are bundled with a shell. MongoDB does not ship with a graphical user interface (MongoDB, 2014d) although there are a number of third-party GUI tools available. During the implementation phase of the project, the support for open source was less than satisfactory but this is bound to change (MongoDB, 2014a).

Due to the fact that the solution required data be aggregated from disparate sources (VCF and Progeny) and with the lack of join support, further investigation into MongoDB was abandoned at this point particularly as the C# driver performance was not satisfactory.

Database Design

The database design was developed with the model-first approach in Entity Framework. Model-first allows the developer to use a designer to create the database schema. The model is defined by the file format *.EDMX* based on XML. DDL (*Database Definition Language*) is generated from the model which creates the database schema when executed against SQL Server.

One of the main drawbacks of model-first is the inability to include some database objects in the designer for example, *non-clustered indexes* and user-defined functions. Adding the syntax directly to DDL results in in the syntax getting overwritten when the model is updated. Therefore certain types of database objects not supported by model-first should be included using the extensibility features in the .EDMX stack to influence the DDL or manage operations in a separate script (Fancey, 2010).

The conceptual model (Figure G.1) was designed by following third normal form (3NF) principles to reduce redundancy. 3NF builds on 1NF and 2NF to organize database records, in other words 1NF and 2NF involve the removal of duplicates and creation of relationships. 3NF indicates the record must conform to 2NF and that all columns in the table are dependent on and the primary key (Chapple, 2014). The database tables are described in Appendix G.

5.4.5 Performance

This section summarises performance best practices considerations and recommendations applied during development. The first load retrieving all results from a data file containing 524,906 total

entries took 832ms[2]. Exporting the dataset of 410,485 entries took 1m 25s. Future development would involve testing the scalability with larger datasets.

General considerations

Poorly formed queries that use up database server resources or database transactions that lock tables can greatly affect a systems availability.

In general, performing *SELECT ** queries (selecting all results) on a dataset is not considered good practice particularly as the amount of records scale. This forms a firm reason why data should be either adequately filtered and/or an upper limit imposed if too many results are returned. *COUNT* (counting records) and *DISTINCT* (selecting unique) are considered expensive queries, and therefore optimization should be performed if they appear to be incurring a performance hit. Only the required data should be queried for.

Poorly performing databases can often be attributed to lack of or incorrectly configured indexes. Indexes prevent contention of IO resources and speed up the retrieval of rows by keys stored in a B-tree structure on disk. There are two main types of indexes applied to tables and views: *Clustered* and *non-clustered* indexes. Clustered indexes are typically unique values like primary keys. Non-clustered indexes are more suitable for read-only environments. Indexes are typically row based although in some cases they can be column-based. For the purpose of this project, row-based indexing is sufficient.

As a general rule of thumb clustered indexes should be applied to all primary key columns and additional non-clustered indexes should be created on columns commonly used in queries and *where* clauses. The order of the columns matter and the query performance can be verified with the execution plan (a feature of SQL Server Management Studio to verify query execution) (Microsoft, 2014a). Statistics of query optimizations are collected by SQL Server. Out-of-date statistics can cause degradation of query performance. SQL Server maintains statistics of query execution plans automatically, unless the settings have been altered (Microsoft, 2014o).

Entity Framework supports different techniques for loading data (eager and lazy). Although the details will not be discussed here, it is important to be aware that care should be taken with lazy loading because related records connected by navigation properties might be inadvertently loaded

[2] This metric was tested during development, refer to Appendix D for system resources.

thus causing a performance hit (known as the *n+1* problem) (Microsoft, 2014c). Merge options specify how objects loaded into the database context are merged with existing objects. Setting the merge option to *NoTracking* in Entity Framework can offer a performance benefit for large read-only data sets (Microsoft, 2014d).

During development, the *CommandTimeout* property was extended in the instantiation of the database context to ensure that on first load there was sufficient time to create the query plan cache. The query plan cache is managed by Entity Framework which can speed up subsequent queries (Obando & Dettinger, 2012).

Applying the *ReadUncommitted* transaction level should be applied to read-only data to prevent locks. Database locks can occur due to a contention of system resources. The concept of *isolation* in database technology determines transaction integrity. By changing the isolation from the default *ReadCommitted* to *ReadUncommitted* can lead to "dirty reads" but because the data in question is read-only, the level can be relaxed (Microsoft, 2014h). The isolation level (Figure 5.6) should be set per database operation within Entity Framework, and not directly to the database. This can be achieved by wrapping a query in a *TransactionScope* object (Microsoft, 2014m).

```
using (var scope = new TransactionScope(TransactionScopeOption.Required,
      new TransactionOptions() { IsolationLevel = IsolationLevel.ReadUncommitted
      }))
      {
          IQueryable<T> toReturn = query;
          scope.Complete();
          return toReturn;
      });
```

Figure 5.6 – Setting the database isolation level in code with Entity Framework.

As a general rule of thumb, if anything can be pre-processed prior to run time, it can be considered a performance optimization. For example if data can be transformed offline, this can greatly impact the performance of the application. Therefore by creating a database design that contains mainly the required information with input fields from data sources adequately parsed, saves the application from doing unnecessary parsing and calculations of data at runtime. The difference between selecting IQueryable versus IEnumerable when iterating data has an effect on performance by offloading processor to the server versus in memory, as does unnecessary casting of objects *ToList*(). Any code using the `dynamic` keyword, or the Dynamic Language Runtime (DLR) is not going to be highly optimized (Watson, 2014). Generally String.Split() and other string

operations are faster than regular expression pattern matching, although there are some circumstances where compiled regular expressions would be faster.

6 Conclusion

NGS technologies have already entered the mainstream, with modern massively parallel platforms such as the Genome Analyzer (*Illumina*), ABI SOLiD (earlier *Applied Biosystems* now *Life Technologies*) and 454 (*Roche*) providing an unprecedented increase in DNA sequencing throughput orders of magnitude higher than Sanger sequencing methods (Simpson, Wong, Jackman, Schein, & Jones, 2009).

Best practices in solving problems in optimal read mapping algorithms are still under development. Sequencing technologies are still rapidly evolving with new methods constantly emerging. Bottlenecks suffered in current methods are often as a result of ever increasing data volumes and sequencers supporting longer read lengths (Liao et al., 2013).

Despite NGS being considered a cheap way of generating vast amounts of sequence data in a single experiment, it can still be quite expensive for small labs working with non-model organisms. Therefore it is important to evaluate that the methodology in place will answer the relevant biological questions. Additionally, labs may need to consider their IT infrastructure due to the sheer amounts of data produced in order to efficiently administer and reuse the data.

The developed database solution can be used for designing new experiments, and thus if the experiment is well designed, it can better answer biological questions and save money from conducting further experiments.

The process of the implementation phase involved selecting a suitable database platform. SQL Server relational database was selected over MongoDB NoSQL as a database storage solution due to the limiting factors encountered with MongoDB including poor driver performance and challenges in creating a data model based on related data.

In my opinion there is much hype around the NoSQL movement where vendors boldly claim typically in a marketing context, to outperform relational database and market features they claim to be ‘novel’ to NoSQL although such features may be already supported in relational database to some degree. NoSQL is still a rapidly evolving area and constantly gaining popularity. Relational database is a mature and trusted technology area where vendors continue to innovate. There is no straight forward answer on whether to adopt a relational database or NoSQL solution, and the answer depends greatly on the problem domain at hand. Despite relational database having

negative connotation about being outdated in terms of scalability and performance, in my opinion it greatly depends on the data model, scale and a myriad of other factors.

The experience could have been different if having tried different 'flavours' of NoSQL, integrating third party products on top of MongoDB to search and index data, or if the scope allowed for more time and effort to be put into investigating the bottlenecks. NoSQL still remains appealing for its schemaless nature, allowing data models to respond more rapidly to change (Isaacson, 2010).

Existing SNP databases designed to search on chromosomal location were reviewed to get an idea of what solutions already existed in the problem domain. A few such examples existed for searching SNPs that allowed search by location and/or filtering by population (e.g. SnpSift, SNPPer, PupaSuite and Slicer) although there was not any evidence of integrations with Progeny.

I would have liked to have further investigated the integration possibilities of SnpSift to identify its potential to incorporate into a variant analysis pipeline. SnpSift currently addresses part of the problem with its SNP filtering capabilities based on mandatory VCF fields and metadata. It is technically feasible to call Java from C#/.NET. One caveat is that SnpSift relies on the presence of SnpEff annotations from the same suite of tools (Cingolani et al., 2012).

The completion of this thesis completed the first release of this variation analysis solution enabling researchers to query and export variation data filtered on populations, habitat, patch and other metadata based on datasets prepared by the MRG, University of Helsinki. As VCF is a standardised format (Danecek et al., 2011), the application has the possibility to be applied to studying variation filtered by population metadata in other organisms. The project will be handed over to the MRG where further releases are planned for example, including a wider range of datasets and integrations with annotation data (e.g. searching Ensemble).

Glossary

Abstract class

A type of class that cannot be instantiated. Abstract classes provide a layer of abstraction where a *signature* for a class's properties and behaviours can be defined.

Action method

In ASP.NET MVC, *action methods* are methods that are invokable via the web from the controller.

AJAX (Asynchronous JavaScript and XML)

Interrelated web development technologies to create asynchronous web applications.

Anonymous type

In C#, an *anonymous type* is a way to create an inline type without explicitly defining it first. Commonly used with *select* clauses in Linq queries.

ASP.NET MVC

Web-based development framework from Microsoft using the MVC pattern.

BAM

The binary version of *SAM*.

Base-calling

The process of assigning bases to sensor data collected during sequencing experiments (Ledergerber & Dessimoz, 2011).

cDNA (Complementary DNA)

DNA produced by reverse transcription of mRNA.

ChIP-seq

NGS method for studying protein interactions and histone modifications combining chromatin immunoprecipitation (ChIP) (Head et al., 2014).

Class

A programming construct that acts as a template for building specific types of objects. Classes can be *instantiated* meaning an instance of an object can be created. Classes are extensible, they can *inherit* from a base class.

CNVs (copy-number variations)

A form of SV incorporating copied segments of sequence (duplication) and associated loss of genetic material (deletion) (Eichler, 2008).

Coding

Strand of DNA that codes for a protein product.

Controller

In ASP.NET MVC, a *controller* is a class that handles incoming requests by mapping URLs to methods.

Coverage

Often used synonymously with read depth, *coverage* is the average number of reads a region has been sequenced (Harbers & Kahl, 2012).

*CSV (*Comma/character separated values*)*

Flat text file format for storing tabular data.

DataTable

An open source data grid control.

Deferred execution

The evaluation of a coding expression is delayed until its *realized* value is requested (Microsoft, 2014g).

Denormalisation

Denormalisation is the process of database optimisation by adding redundancy (Shin & Sanders, 2006).

Dispersal

Simply speaking, dispersal is the movement of organisms away from their place of birth (Levin, 2009).

DNA (Deoxyribonucleic acid)

DNA is a biomolecule consisting of four bases, represented by the letters A, C, G and T which acts as a carrier of genetic information.

Ecological genomics

Study of genomics in natural population (Levin, 2009).

Entity Framework

ORM library from Microsoft which provides a layer of abstraction between the database and application logic (Microsoft, 2014b).

ETL (Extract, Transform, Load)

Database process where data is *extracted* from various data sources, *transformed* into an appropriate structure for querying and *loaded* into the target e.g. a database (Horváth, 2013).

Eukaryote

Organism with a true nucleus.

Exon

The coding region of a gene.

Extension method

Type of method syntax which allows methods to be called on types inline which are a special type of static method commonly seen in e.g. *Linq*.

Fitness

In Darwinian natural selection, fitness is a measure of an organism's capacity to survive and reproduce.

Foreign key

A type of relational database constraint that creates relationships between tables in a database. Unlike primary keys, foreign keys are not required to be unique.

Fragmented landscape

Discontinuities in an organism's habitat resulting in fragmentation of a population (Levin, 2009).

Gene flow

Genetic exchange between populations.

Genetic drift

The process of change in the allele frequencies of a population caused by random sampling (Ahluwalia, 2009).

Genetic variation

Genetic variation consists of SNP and indel polymorphisms, structural variation and chromosome number alterations between individuals.

Genotype calling

An application of NGS that occurs after mapping reads to a reference sequence. In genotype calling, the genotype is determined for each individual at each site (Nielsen et al., 2011).

Genotype

The genetic constitution of an organism.

Genotyping

The process of determining the *genotype* of an individual (Coriell Institute for Medical Research, 2014).

Habitat patch

Continuous range of environment conditions under which a species may survive and reproduce (I Hanski, 2005)

Habitat patch network

Collection of spatially distinct habitat patches connected by linear element (structural definition) or linked by a flow of individuals from patch to patch (functional definition) (Gutzwiller, 2002).

Haploid

A type of cell containing a single set of chromosomes involved in sexual reproduction (Clark, 2009).

Heterozygous

Refer to *zygosity*.

Homozygous

Refer to *zygosity*

HTS (High throughput sequencing)

Synonymous with NGS.

IEnumerable

In C#, IEnumerable is an interface to a collection that exposes an iterator.

Indel

Indel is an abbreviation for insertion/deletion mutation, the length of which can vary.

Intergenic region

DNA sequence preceding genes that have no proven function.

Interface

In C#, an interface defines a contact which governs the allowed syntax for a class. By separating the definition of objects from their implementation increases flexibility in altering the underlying implementation and improves testability.

Insert size

The fragments between the adapters in a library.

IQueryable

In C#, *IQueryable* is an interface that inherits from *IEnumerable* to evaluate queries against external data sources.

Join

In SQL, a *JOIN* is a clause that combines records from two or more tables.

JSON (JavaScript Object Notation)

An open-standard, human readable transport format offering a lighter-weight alternative to XML. JSON is a popular format for passing AJAX between client and server (Json.org, n.d.).

Landscape

A human-defined area of the natural terrestrial world (Levin, 2009).

LD (Linkage disequilibrium)

The non-random association of alleles at two or more loci and is commonly used as a measure for correlations between linked loci (Peng, Amos, & Kimmel, 2012).

Library (sequencing)

A set of nucleic acid fragments that has been processed and is ready for sequencing (Head et al., 2014).

Library (code)

A method for encapsulating code in a reusable package. In Windows-based systems, libraries typically contains the .DLL file extension.

LIMS (Laboratory Information Management System)

Software to support information management of data and operations in laboratory environments (Sapio Sciences, 2010).

Metapopulation

A group of spatially separated populations linked via dispersal (Levin, 2009).

MRG (Metapopulation research group)

Research group at the University of Helsinki researching species inhabiting fragmented landscapes (University of Helsinki, 2006).

mRNA (messenger RNA)

RNA molecule that carries genetic information from DNA during transcription.

Mutation

Permanent change in the DNA sequence of the genome. Considered the ultimate source of genetic variation in the form of new alleles (Clark, 2009).

MVC (model-view-controller)

A common development paradigm for the isolation of concerns.

Natural selection

Process where phenotypic traits in individuals in a population become more or less common based on inherited traits of the differential reproductive success (Levin, 2009).

Namespace

Mechanism for avoiding conflicts in element naming. In .NET namespaces are frequently used to organise classes.

NGS (Next Generation Sequencing)

The current state of DNA sequencing technology that is cost effective compared to earlier sequencing methods.

Non-coding

Strand of DNA that does not code for a protein product.

NoSQL

Schemaless, non-relational database technology movement that began in 2009 (Nosql-databases.org, 2014).

Object model

In software development, the set of interactions required to interact with an external source. In this case, the object model is commonly used to refer to .NET libraries included from other sources.

ORM (object relational mapper)

Software that provides a layer of abstraction between the database and application logic (Fancey, 2010).

Paired-end read

Methodology used in sequencing where both ends of a DNA fragment are sequenced (as opposed to a single-end in a single-end read) coupled with distance information (R. Li et al., 2009).

Phred quality score

Type of quality score used in NGS experiments during base-calling denoted by:

$$Q_{Phred} = -10\log_{10} P(\text{error})$$

(Nielsen et al., 2011)

Point mutation

A mutation that results from the replacement of a single nucleotide base with another. *Indels* are a type of point mutation (Clark, 2009).

Polymorphism

In relation to genetics, polymorphism describes multiple forms of a DNA sequence, gene or chromosome that can exist in an individual or group of individuals (Ahluwalia, 2009).

Population genetics

A discipline of evolutionary biology that studies the variation between individuals and populations (Levin, 2009).

Primary key

A type of relational database constraint that uniquely identifies each row in a table.

Progeny

Progeny Lab is a proprietary LIMS for the centralised management of genetic data (Progeny Software, 2014).

Project

In software development, a Visual Studio project refers to a logical grouping of artefacts used to build into an output type (e.g. an executable). Projects are contained within Visual Studio *solutions*.

Pyrosequencing

Method of DNA sequencing based on the *Sequencing by synthesis* principal. Roche 454 is a Pyrosequencing platform (Metzker, 2010).

Quality score

In the context of NGS, commonly referred to as a *Phred* quality score.

RDBMS (Relational database management system)

Type of database based on the relational model.

Read depth

See *coverage*.

Recombination

The formation of new combinations of genetic material for example, by crossing-over between homologous chromosomes or the breaking and joining of DNA strands (Clark, 2009).

Relational model

Database model proposed by E.F. Codd. Data in the relational model is represented in terms of tuples and relations (Codd, 1970).

Resequencing

One of the more popular applications of NGS used for example to determine genomic variation in relation to a reference genome (Flicek & Birney, 2010).

RNA (Ribonucleic acid)

Molecule involved in protein synthesis.

RNA-seq

NGS technique for RNA analysis through cDNA sequencing (Corney, 2012).

Root-mean-square errors (RMSE)

The square root of the Mean-squared Error (MSE). MSE is a measure of how close a fitted line is to data points. RMSE represents the standard variation (Vernier, 2011).

SAM

Sequence Alignment/Map format is a file format for the storage of read alignments against a reference sequence (H. Li et al., 2009).

Sanger sequencing

Before the wide adoption of NGS, Sanger sequencing was the most common DNA sequencing method (R. Li et al., 2009). Sanger sequencing is based on a capillary sequencing technology, and is still widely used in small-scale projects.

Sequencing

In the context of this thesis, *sequencing* refers to DNA sequencing; the process of determining the exact order of the bases from DNA (Altmann et al., 2012).

Sequencing-by-synthesis

Sequencing-by-synthesis (SBS) is a method of DNA sequencing based on clonal cluster sequencing technology. SBS falls into two schools: Terminator or Pyrosequencing based (Harbers & Kahl, 2012).

Service class

Typical design pattern that fetches data from a given data source (e.g. a repository), processes the information and returns it the caller. This provides a layer of abstraction between the controller and the database context.

Short read

Short read sequences are the raw sequence results produced from NGS.

SNP (single nucleotide polymorphism)

Genetic variant where one nucleotide is substituted for another.

SNP calling

Refer to *variant calling*.

Solution

In software development, a Visual Studio solution acts as a container for projects. The solution file is defined by a *.sln* extension.

Splice junction

Site of a former intron in a mature mRNA during *splicing* (a stage of processing mRNA where introns are removed) (ZFIN, 2014).

*SQL (***Structured Query Language***)*

Query language used to access data in relational database systems.

SQL Server

Microsoft's proprietary relational database software.

Static

In C#, static variables exist for the lifetime that the application runs i.e. an instance of a static object cannot be created.

Strongly-typed

Strongly-typed is a term used to describe programming languages that impose strict restrictions on the declaration and assignment of data types and flagging an error if violation occurs (Microsoft, 2014n).

Substitution

Substitution refers to the replacement of a single nucleotide base with another.

SV (Structural Variant)

Type of variant typically > 1 *kb* encompassing CNVs and genomic arrangements (Feuk, Carson, & Scherer, 2006).

Table

An entity in relational databases consisting of a collection of related data structured by columns and rows.

TransactionScope

In C#, a *TransactionScope* object (from the *System.Transactions.TransactionScope* namespace) provides the ability to call database operations within a database transaction which comprises a logical unit of work. This provides more control over the database operation e.g. preventing database locks (Microsoft, 2014m).

UI (User interface)

The aspects of a computer system or program in which the user interacts with.

User-defined Function

Custom function written in SQL that encapsulates logic, accepts input parameters and returns data (Microsoft, 2014p).

UTR (untranslated region)

Either of two sections on each side of a coding sequence on a strand of mRNA (Clark, 2009) .

Variant calling

Identification of biological variants from aligned sequence data in the genomes of populations of interest (Altmann et al., 2012).

VCF (Variant Call Format)

Tab-delimited file format for storage of variation data (Danecek et al., n.d.).

View

In ASP.NET MVC, a *view* represents the components required to display in the UI.

View model

In ASP.NET MVC, a view model *models* the properties required by the view.

Visual Studio

An IDE (integrated development environment) from Microsoft.

Zygosity

In diploid organisms, homozygous refers to having identical alleles for two sets of homologous chromosomes and heterozygous refers to having different alleles for two sets of homologous chromosomes. (Ahluwalia, 2009)

References

Abecasis, G. R., Altshuler, D., Auton, A., Brooks, L. D., Durbin, R. M., Gibbs, R. A., … McVean, G. A. (2010). A map of human genome variation from population-scale sequencing. *Nature*, *467*(7319), 1061–73. doi:10.1038/nature09534

About the 1000 Genomes Project. (n.d.). Retrieved June 09, 2013, from http://www.1000genomes.org/about

Ahluwalia, K. B. (2009). *Genetics*. Daryaganj, Delhi, IND: New Age International. Retrieved from http://site.ebrary.com/lib/uniturku/docDetail.action?docID=10318687

Ahola, V., Lehtonen, R., Somervuo, P., Salmela, L., Koskinen, P., Rastas, P., … Hanski, I. (2014). The Glanville fritillary genome retains an ancient karyotype and reveals selective chromosomal fusions in Lepidoptera. *Nature Communications*, *5*, 4737. doi:10.1038/ncomms5737

Altmann, A., Weber, P., Bader, D., Preuss, M., Binder, E. B., & Müller-Myhsok, B. (2012). A beginners guide to SNP calling from high-throughput DNA-sequencing data. *Human Genetics*, *131*(10), 1541–54. doi:10.1007/s00439-012-1213-z

Altshuler, D. M., Gibbs, R. A., Peltonen, L., Dermitzakis, E., Schaffner, S. F., Yu, F., … McEwen, J. E. (2010). Integrating common and rare genetic variation in diverse human populations. *Nature*, *467*(7311), 52–8. doi:10.1038/nature09298

Andrew, R. L., Bernatchez, L., Bonin, A., Buerkle, C. A., Carstens, B. C., Emerson, B. C., … Rieseberg, L. H. (2013). A road map for molecular ecology. *Molecular Ecology*, *22*(10), 2605–26. doi:10.1111/mec.12319

Andrews, C. A. (2010). Natural Selection, Genetic Drift, and Gene Flow Do Not Act in Isolation in Natural Populations. *Nature Education Knowledge*. doi:3(10):5

Benjamini, Y., & Speed, T. P. (2012). Summarizing and correcting the GC content bias in high-throughput sequencing. *Nucleic Acids Research*, *40*(10), e72. doi:10.1093/nar/gks001

Bukharin. (2014). Grid.Mvc - Home. *CodePlex*. Retrieved September 29, 2014, from https://gridmvc.codeplex.com/

Burrows-Wheeler Aligner. (n.d.). Retrieved August 03, 2013, from http://bio-bwa.sourceforge.net/

Carlin, J. L. (2011). Mutations Are the Raw Materials of Evolution. *Nature Education Knowledge*. doi:3(10):10

Casals, F., Idaghdour, Y., Hussin, J., & Awadalla, P. (2012). Next-generation sequencing approaches for genetic mapping of complex diseases. *Journal of Neuroimmunology*, *248*(1-2), 10–22. doi:10.1016/j.jneuroim.2011.12.017

Chapple, M. (2014). Third Normal Form - Converting Databases to 3NF. *About Technology*. Retrieved October 12, 2014, from http://databases.about.com/od/specificproducts/a/3nf.htm

Charlesworth, B. (2010). Molecular population genomics: a short history. *Genetics Research*, *92*(5-6), 397–411. doi:10.1017/S0016672310000522

Cingolani, P., Patel, V. M., Coon, M., Nguyen, T., Land, S. J., Ruden, D. M., & Lu, X. (2012). Using Drosophila melanogaster as a Model for Genotoxic Chemical Mutational Studies with a New Program, SnpSift. *Frontiers in Genetics*, *3*, 35. doi:10.3389/fgene.2012.00035

Clark, D. P. (2009). *Molecular Biology*. Burlington, MA, USA: Elsevier Science & Technology. Retrieved from http://site.ebrary.com/lib/uniturku/docDetail.action?docID=10408166

Clarke, L., Zheng-Bradley, X., Smith, R., Kulesha, E., Xiao, C., Toneva, I., … Flicek, P. (2012). The 1000 Genomes Project: data management and community access. *Nature Methods*, *9*(5), 459–62. doi:10.1038/nmeth.1974

Codd, E. F. (1970). A relational model of data for large shared data banks. 1970. *M.D. Computing : Computers in Medical Practice*, *15*(3), 162–6. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/9617087

Conde, L., Vaquerizas, J. M., Dopazo, H., Arbiza, L., Reumers, J., Rousseau, F., … Dopazo, J. (2006). PupaSuite: finding functional single nucleotide polymorphisms for large-scale genotyping purposes. *Nucleic Acids Research*, *34*(Web Server issue), W621–5. doi:10.1093/nar/gkl071

Coriell Institute for Medical Research. (2014). What is Genotyping and Expression Profiling? Retrieved October 13, 2014, from http://www.coriell.org/research-services/genotyping-microarray/what-is-genotyping-and-expression-profiling

Corney, D. C. (2012). RNA-seq Using Next Generation Sequencing. *Materials and Methods*, *2*. doi:10.13070/mm.en.2.203

Dalca, A. V, & Brudno, M. (2010). Genome variation discovery with high-throughput sequencing data. *Briefings in Bioinformatics*, *11*(1), 3–14. doi:10.1093/bib/bbp058

Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., … Durbin, R. (n.d.). The Variant Call Format and VCFtools. Retrieved November 14, 2013, from http://vcftools.sourceforge.net/VCF-poster.pdf

Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., … Genomes Project Analysis, G. (2011). The Variant Call Format and VCFtools. *Bioinformatics*, *27*, 2156–2158. Retrieved from http://bioinformatics.oxfordjournals.org/content/27/15/2156.abstract

DePristo, M. A., Banks, E., Poplin, R., Garimella, K. V, Maguire, J. R., Hartl, C., … Daly, M. J. (2011). A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature Genetics*, *43*(5), 491–8. doi:10.1038/ng.806

DevExpress Inc. (2014). Overview - ASP.NET MVC Extensions - Grid View Demo | DevExpress. *DevExpress*. Retrieved September 29, 2014, from http://demos.devexpress.com/MVCxGridViewDemos/Overview

Eichler, E. E. (2008). Copy Number Variation and Human Disease. *Nature Education*, *1*(3), 1. Retrieved from http://www.nature.com/scitable/topicpage/copy-number-variation-and-human-disease-741737

Ekblom, R., & Galindo, J. (2011). Applications of next generation sequencing in molecular ecology of non-model organisms. *Heredity*, *107*(1), 1–15. doi:10.1038/hdy.2010.152

Ellegren, H., & Sheldon, B. C. (2008). Genetic basis of fitness differences in natural populations. *Nature*, *452*(7184), 169–75. doi:10.1038/nature06737

Fancey, J. (2010). Model-First in the Entity Framework 4. *Data Developer Center*. Retrieved September 25, 2014, from http://msdn.microsoft.com/en-us/data/ff830362.aspx

Ferragina, P., & Venturini, R. (2005). Fm-index version 2. Retrieved August 03, 2013, from http://www.di.unipi.it/~ferragin/Libraries/fmindexV2/

Feuk, L., Carson, A. R., & Scherer, S. W. (2006). Structural variation in the human genome. *Nature Reviews. Genetics*, *7*(2), 85–97. doi:10.1038/nrg1767

Flicek, P., & Birney, E. (2010). Sense from sequence reads : methods for alignment and assembly, *6*(11). doi:10.1038/NMETH.1376

Freeman, A. (2013). *Pro ASP.NET MVC 5 (Expert's Voice in ASP.Net)* (5th ed.). Apress. Retrieved from http://amazon.com/o/ASIN/1430265299/

Gutzwiller, K. (2002). *Applying landscape ecology in biological conservation*. New York: Springer.

Hanski, I. (1999). *Metapopulation ecology*. Oxford New York: Oxford University Press.

Hanski, I. (2005). *The shrinking world: ecological consequences of habitat loss*. International Ecology Institute. Retrieved from http://books.google.fi/books?id=dCtFAQAAIAAJ

Hanski, I. a. (2011). Eco-evolutionary spatial dynamics in the Glanville fritillary butterfly. *Proceedings of the National Academy of Sciences of the United States of America*, *108*(35), 14397–404. doi:10.1073/pnas.1110020108

Harbers, M., & Kahl, G. (2012). *Tag-based Next Generation Sequencing*. Somerset, NJ, USA: Wiley-Blackwell. Retrieved from http://site.ebrary.com/lib/uniturku/docDetail.action?docID=10630647

Head, S. R., Komori, H. K., LaMere, S. a, Whisenant, T., Van Nieuwerburgh, F., Salomon, D. R., & Ordoukhanian, P. (2014). Library construction for next-generation sequencing: overviews and challenges. *BioTechniques*, *56*(2), 61–4, 66, 68, passim. doi:10.2144/000114133

Hedberg, J. (2013). dsDNA-3D-stylized. Retrieved from http://www.jameshedberg.com/lettera/2010/06/14/dna-model-1/

Horváth, Z. (2013). Extract Transform Load (ETL) - TechNet Articles - United States (English) - TechNet Wiki. *Technet*. Retrieved October 13, 2014, from http://social.technet.microsoft.com/wiki/contents/articles/4475.extract-transform-load-etl.aspx

IBBL. (n.d.). Single Nucleotide Polymorphism. Retrieved from http://www.ibbl.lu/wp-content/uploads/2012/07/SNPs.jpg

Illumina. (2014). Sequencing Systems | Sequencer Comparison Table | Illumina. Retrieved September 27, 2014, from http://systems.illumina.com/systems/sequencing.ilmn

Isaacson, C. (2010). SQL or NoSQL? How to Choose the Right Database for Your Application - Database Trends and Applications. *Database Trends and Applications*. Retrieved September 30, 2014, from http://www.dbta.com/Editorial/Trends-and-Applications/SQL-or-NoSQL-How-to-Choose-the-Right-Database-for-Your-Application-71240.aspx

Json.org. (n.d.). JSON. Retrieved October 13, 2014, from http://json.org/

Kim, D., Pertea, G., Trapnell, C., Pimentel, H., Kelley, R., & Salzberg, S. L. (2013). TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology*, *14*(4), R36. doi:10.1186/gb-2013-14-4-r36

Kvist, J. (2014). *Functional genomics of the Glanville fritillary butterfly*. University of Helsinki. Retrieved from http://urn.fi/URN:ISBN:978-951-51-0247-8

Ledergerber, C., & Dessimoz, C. (2011). Base-calling for next-generation sequencing platforms. *Briefings in Bioinformatics*, *12*(5), 489–97. doi:10.1093/bib/bbq077

Lederman, R. (2013). A random-permutations-based approach to fast read alignment. *BMC Bioinformatics*, *14 Suppl 5*(Suppl 5), S8. doi:10.1186/1471-2105-14-S5-S8

Levin, S. (2009). *Princeton Guide to Ecology*. Princeton, NJ, USA: Princeton University Press. Retrieved from http://site.ebrary.com/lib/uniturku/docDetail.action?docID=10405141

Li, H., & Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics (Oxford, England)*, *25*(14), 1754–60. doi:10.1093/bioinformatics/btp324

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., … Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, *25*(16), 2078–9. doi:10.1093/bioinformatics/btp352

Li, H., & Homer, N. (2010). A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, *11*(5), 473–83. doi:10.1093/bib/bbq015

Li, H., Ruan, J., & Durbin, R. (2008). Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research*, *18*(11), 1851–8. doi:10.1101/gr.078212.108

Li, R., Li, Y., Fang, X., Yang, H., Wang, J., Kristiansen, K., & Wang, J. (2009). SNP detection for massively parallel whole-genome resequencing. *Genome Research*, *19*(6), 1124–32. doi:10.1101/gr.088013.108

Liao, Y., Smyth, G. K., & Shi, W. (2013). The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic Acids Research*, *41*(10), e108. doi:10.1093/nar/gkt214

Lozinski, D. (2013). C# .Net: Fastest Way to Read Text Files - The Curious Consultant. *The Curious Consultant*. Retrieved September 28, 2014, from http://cc.davelozinski.com/c-sharp/fastest-way-to-read-text-files

Masters Emison, J. (2014). 2014 State Of Database Tech: Think Retro - InformationWeek. *InformationWeek*. Retrieved September 24, 2014, from http://www.informationweek.com/software/information-management/2014-state-of-database-tech-think-retro/d/d-id/1114186

Matos, A. (2014). Datatables.Mvc. *GitHub*.

Mattila, A. L. K., Duplouy, A., Kirjokangas, M., Lehtonen, R., Rastas, P., & Hanski, I. (2012). High genetic load in an old isolated butter fly population, *109*(37). doi:10.1073/pnas.1205789109/-/DCSupplemental.www.pnas.org/cgi/doi/10.1073/pnas.1205789109

Metzker, M. L. (2010). Sequencing technologies - the next generation. *Nature Reviews. Genetics*, *11*(1), 31–46. doi:10.1038/nrg2626

Microsoft. (2014a). Clustered and Nonclustered Indexes Described. *Developer Network*. Retrieved September 29, 2014, from http://msdn.microsoft.com/en-us/library/ms190457.aspx

Microsoft. (2014b). Entity Framework. *Data Developer Center*. Retrieved September 25, 2014, from http://msdn.microsoft.com/en-us/data/ef.aspx

Microsoft. (2014c). Entity Framework Loading Related Entities. *Developer Network*. Retrieved September 29, 2014, from http://msdn.microsoft.com/en-us/data/jj574232.aspx

Microsoft. (2014d). Entity Framework No-Tracking Queries. *Developer Network*. Retrieved September 29, 2014, from http://msdn.microsoft.com/en-us/data/jj556203.aspx

Microsoft. (2014e). Introduction to the C# Language and the .NET Framework. *Developer Network*. Retrieved September 25, 2014, from http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

Microsoft. (2014f). LINQ (Language-Integrated Query). *Developer Network*. Retrieved September 25, 2014, from http://msdn.microsoft.com/en-us/library/bb397926.aspx

Microsoft. (2014g). Query Execution. *MSDN*. Retrieved October 13, 2014, from http://msdn.microsoft.com/en-us/library/vstudio/bb738633(v=vs.100).aspx

Microsoft. (2014h). SET TRANSACTION ISOLATION LEVEL (Transact-SQL). *Developer Network*. Retrieved September 29, 2014, from http://msdn.microsoft.com/en-us/library/ms173763.aspx

Microsoft. (2014i). SQL Server Integration Services. *Technet*. Retrieved September 24, 2014, from http://technet.microsoft.com/en-us/library/ms141026.aspx

Microsoft. (2014j). SSIS Package Essentials. *Technet*. Retrieved September 30, 2014, from http://technet.microsoft.com/en-us/library/cc280511(v=sql.120)

Microsoft. (2014k). System.Linq Namespace (). *MSDN*. Retrieved October 15, 2014, from http://msdn.microsoft.com/en-us/library/system.linq.aspx

Microsoft. (2014l). Transactions. *Technet*. Retrieved October 13, 2014, from http://technet.microsoft.com/en-us/library/aa213068(v=sql.80).aspx

Microsoft. (2014m). TransactionScope Class (System.Transactions). *MSDN*. Retrieved October 13, 2014, from http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope(v=vs.110).aspx

Microsoft. (2014n). Types (C# Programming Guide). *MSDN*. Retrieved October 13, 2014, from http://msdn.microsoft.com/en-us/library/ms173104.aspx

Microsoft. (2014o). UPDATE STATISTICS (Transact-SQL). *Developer Network*. Retrieved September 29, 2014, from http://msdn.microsoft.com/en-us/library/ms187348.aspx

Microsoft. (2014p). User-Defined Functions. *MSDN*. Retrieved October 13, 2014, from http://msdn.microsoft.com/en-us/library/ms191007.aspx

MongoDB. (2014a). Admin UIs. *MongoDB Ecosystem*. Retrieved September 28, 2014, from http://docs.mongodb.org/manual/core/document/

MongoDB. (2014b). C# and .NET MongoDB Driver. *MongoDB Ecosystem*. Retrieved September 28, 2014, from http://docs.mongodb.org/ecosystem/drivers/csharp/

MongoDB. (2014c). Database References — MongoDB Manual 2.6.4. *MongoDB Manual 2.6*. Retrieved September 28, 2014, from http://docs.mongodb.org/manual/reference/database-references/

MongoDB. (2014d). Documents — MongoDB Manual 2.6.4. *MongoDB Manual 2.6*. Retrieved September 24, 2014, from http://docs.mongodb.org/manual/core/document/

MongoDB. (2014e). GridFS — MongoDB Manual 2.6.4. *MongoDB Manual 2.6*. Retrieved September 24, 2014, from http://docs.mongodb.org/manual/core/gridfs/

MongoDB Inc. (2014). MongoDB C# Driver API Documentation -. *MongoDB C# Driver API Documentation*. Retrieved September 28, 2014, from http://api.mongodb.org/csharp/current/

Morin, P. a., Luikart, G., & Wayne, R. K. (2004). SNPs in ecology, evolution and conservation. *Trends in Ecology & Evolution*, *19*(4), 208–216. doi:10.1016/j.tree.2004.01.009

Multisample SNP Calling. (2010). Retrieved November 21, 2013, from http://samtools.sourceforge.net/mpileup.shtml

Nakamura, K., Oshima, T., Morimoto, T., Ikeda, S., Yoshikawa, H., Shiwa, Y., … Kanaya, S. (2011). Sequence-specific error profile of Illumina sequencers. *Nucleic Acids Research*, *39*(13), e90. doi:10.1093/nar/gkr344

Nielsen, R., Hellmann, I., Hubisz, M., Bustamante, C., & Clark, A. G. (2007). Recent and ongoing selection in the human genome. *Nature Reviews. Genetics*, *8*(11), 857–68. doi:10.1038/nrg2187

Nielsen, R., Paul, J. S., Albrechtsen, A., & Song, Y. S. (2011). Genotype and SNP calling from next-generation sequencing data. *Nature Reviews. Genetics*, *12*(6), 443–51. doi:10.1038/nrg2986

Norrgard, K. (Write S. R., & Schultz, J. (Write S. R. (2008). Using SNP Data to Examine Human Phenotypic Differences. *Nature Education*. Retrieved September 14, 2013, from http://www.nature.com/scitable/topicpage/using-snp-data-to-examine-human-phenotypic-706#

Nosql-databases.org. (2014). NOSQL Databases. Retrieved September 24, 2014, from http://nosql-database.org/

Obando, D., & Dettinger, E. (2012). Performance Considerations for EF 4, 5, and 6. *MSDN*. Retrieved October 12, 2014, from http://msdn.microsoft.com/en-us/data/hh949853.aspx

Oracle. (2007). Oracle Timeline. *Profit*, (May), 26–33. Retrieved from http://www.oracle.com/us/corporate/profit/p27anniv-timeline-151918.pdf

Outercurve Foundation. (2014). NuGet Gallery | Home. Retrieved October 11, 2014, from http://www.nuget.org/

Pearson, H. (2006). Genetics: what is a gene? *Nature*, *441*(7092), 398–401. doi:10.1038/441398a

Peng, B., Amos, C. I., & Kimmel, M. (2012). *Forward-Time Population Genetics Simulations : Methods, Implementation, and Applications*. Hoboken, NJ, USA: Wiley-Blackwell. Retrieved from http://site.ebrary.com/lib/uniturku/docDetail.action?docID=10534012

Perdeck, M. (2014). LINQ to CSV library - CodeProject. *Code Project*. Retrieved September 25, 2014, from http://www.codeproject.com/Articles/25133/LINQ-to-CSV-library

Progeny Software. (2014). SNP, Genotype Management Software - Progeny Lab. Retrieved October 13, 2014, from http://www.progenygenetics.com/lab/

Quinn, E. M., Cormican, P., Kenny, E. M., Hill, M., Anney, R., Gill, M., … Morris, D. W. (2013). Development of strategies for SNP detection in RNA-seq data: application to lymphoblastoid cell lines and evaluation using 1000 Genomes data. *PloS One*, *8*(3), e58815. doi:10.1371/journal.pone.0058815

Redman, M. (2008). SQL Server Best Practices – Implementation of Database Object Schemas. *Technet*. Retrieved September 24, 2014, from http://technet.microsoft.com/en-us/library/dd283095(v=sql.100).aspx

Riva, A., & Kohane, I. S. (2004). A SNP-centric database for the investigation of the human genome. *BMC Bioinformatics*, *5*, 33. doi:10.1186/1471-2105-5-33

Samtools. (2013). The Variant Call Format (VCF) Version 4 . 1 Specification The VCF specification, 1–27. Retrieved from http://samtools.github.io/hts-specs/VCFv4.1.pdf

Sapio Sciences. (2010). Laboratory Information Management: So what is a LIMS? *Laboratory Information Management*. Retrieved October 30, 2014, from http://sapiosciences.blogspot.fi/2010/07/so-what-is-lims.html

Seward, J. (n.d.). bzip2 and libbzip2, version 1.0.5. Retrieved November 07, 2013, from http://bzip.org/1.0.5/bzip2-manual-1.0.5.html

Shin, S. K., & Sanders, G. L. (2006). Denormalization strategies for data retrieval from data warehouses. *Decision Support Systems*, *42*(1), 267–282. doi:10.1016/j.dss.2004.12.004

Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., & Jones, S. J. M. (2009). ABySS : A parallel assembler for short read sequence data, 1117–1123. doi:10.1101/gr.089532.108.

SNP Class Definitions. (2005). National Center for Biotechnology Information (US). Retrieved from http://www.ncbi.nlm.nih.gov/books/NBK44488/

SpryMedia. (2014). DataTables | Table plug-in for jQuery. *DataTables*. Retrieved September 28, 2014, from http://datatables.net/

The SAM/BAM Format Specification Working Group. (2014). Sequence Alignment / Map Format Specification, 1–16. Retrieved from http://samtools.github.io/hts-specs/SAMv1.pdf

Twyman, R. (Wellcome T. (2003). Gene structure | The Human Genome. Retrieved October 24, 2013, from http://genome.wellcome.ac.uk/doc_wtd020755.html

University of Helsinki. (2006). Metapopulation Research Group. Retrieved October 30, 2014, from http://www.helsinki.fi/science/metapop/

Watson, B. (2014). Performance Considerations of Class Design and General Coding in .NET - CodeProject. *Code Project*. Retrieved September 25, 2014, from http://www.codeproject.com/Articles/812678/Performance-Considerations-of-Class-Design-and-Gen

VCFtools. (2014). VCFtools Documentation. Retrieved November 21, 2013, from http://vcftools.sourceforge.net/docs.html

Vera, J. C., Wheat, C. W., Fescemyer, H. W., Frilander, M. J., Crawford, D. L., Hanski, I., & Marden, J. H. (2008). Rapid transcriptome characterization for a nonmodel organism using 454 pyrosequencing. *Molecular Ecology*, *17*(7), 1636–47. doi:10.1111/j.1365-294X.2008.03666.x

Vernier. (2011). What are Mean Squared Error and Root Mean Squared Error? > Vernier Software & Technology. *Tech Info Library*. Retrieved November 11, 2014, from http://www.vernier.com/til/1014/

What Is the HapMap. (n.d.). *International HapMap Project*. Retrieved January 07, 2013, from http://hapmap.ncbi.nlm.nih.gov/whatishapmap.html.en

ZFIN. (2014). ZFIN Glossary. Retrieved October 13, 2014, from http://site.ebrary.com/lib/uniturku/docDetail.action?docID=10405141

Appendices

Appendix A – Application Screen Shots

Data sources section

The *Data Sources* section displays the available data sources for search (Figure A.1).

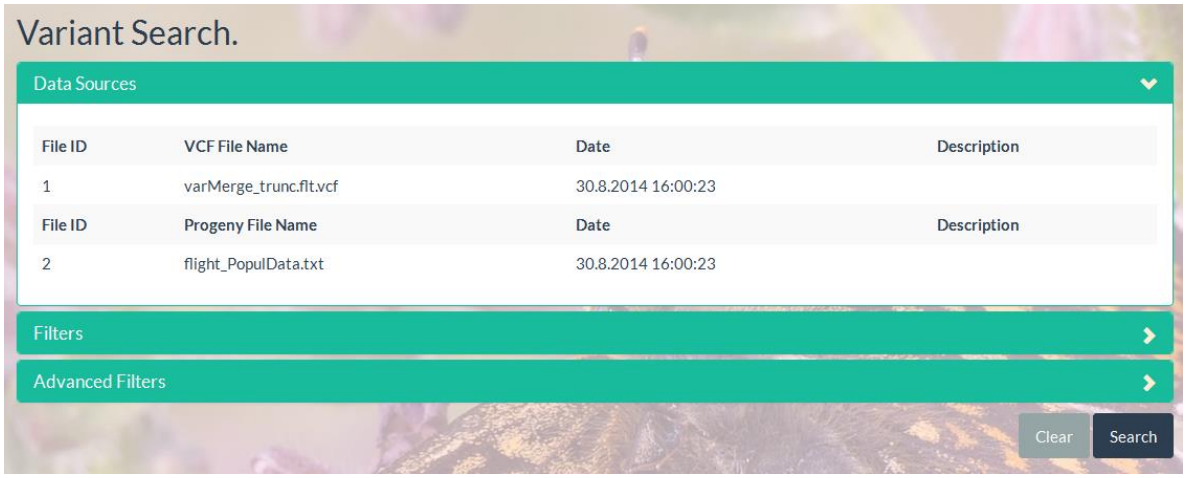


Figure A.1 – Screenshot of the search page showing the *Data Sources* section. Data sources are grouped by file type.

Filters section

The *Filters* section displays the most common filters for search (Figure A.2).

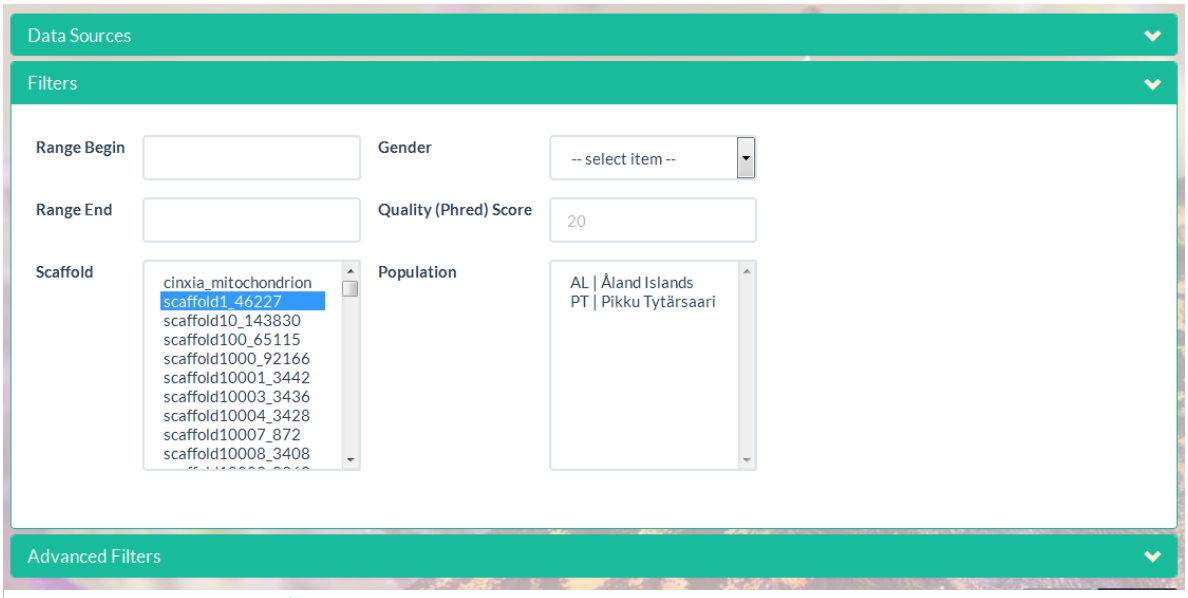


Figure A.1 – Screenshot of the search page showing the *Filters* section. Context-sensitive help is provided when mousing-over column names.

Advanced Filters section

The *Advanced Filters* section displays less common filters for search (Figure A.3).

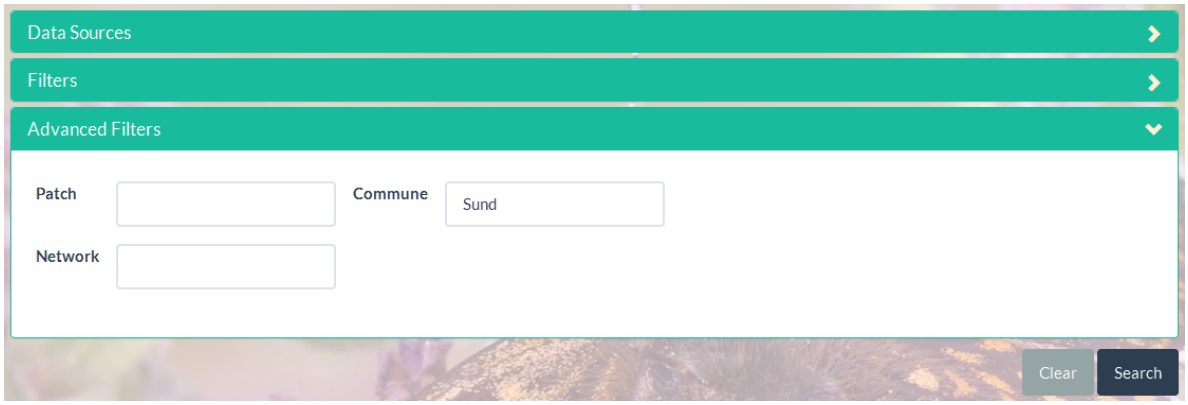


Figure A.3 – Screenshot of the search page showing the *Advanced Filters* section. Text boxes support auto-complete for existing values. Context-sensitive help is provided when mousing over column names.

Search results section

The search results section shows the DataTables data grid populated with search results after the search form is submitted (Figure A.4).

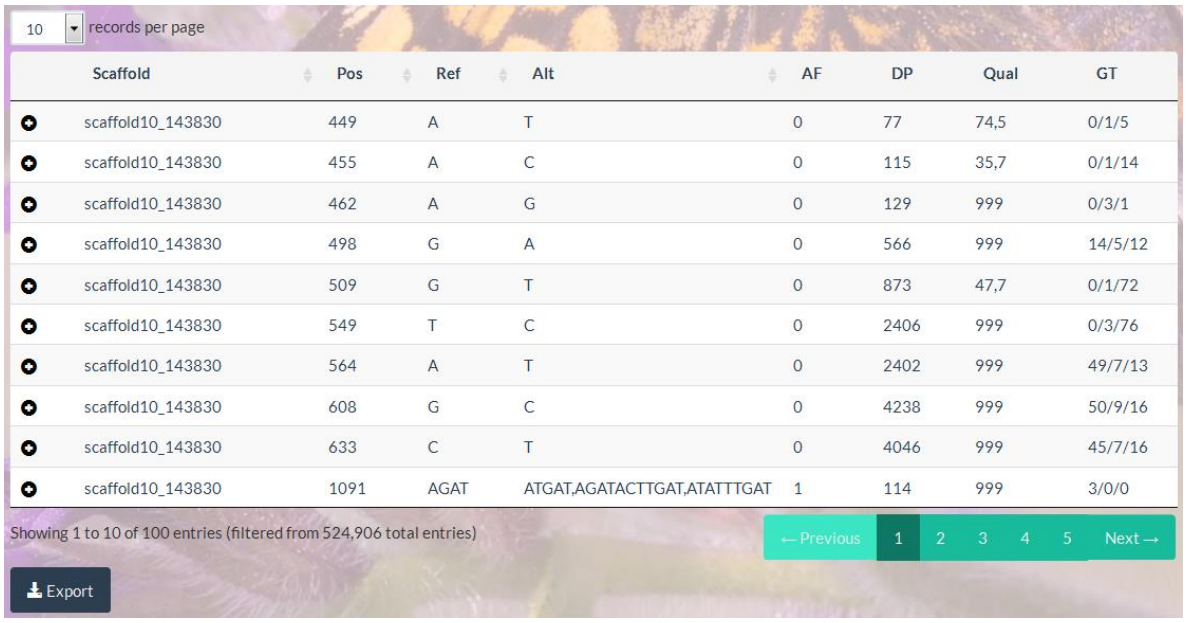


Figure A.4 – Screenshot of the search page showing the search results section. Displays the DataTable control populated with search results. The plus in the left-most column displays the master-detail view. The table footer shows the number of results filtered from the number of total entries on the left. To the right, the pagination control is shown. The number of results can be altered by selecting the drop-down at the top left displaying *n* results per page. Context-sensitive help is provided when mousing-over column names. The export button appears below the data grid allowing the total filtered entries to be exported to CSV.

Appendix B – Data Source Samples

List of data sources

Type	*Size*	*Rows*
VCF	789 MB	524,906
Progeny	14 KB	85 (1 row per individual)

Table B.1 – Files sizes of data sources used during the implementation phase. Single related VCF and Progeny files were used during project implementation.

VCF

##fileformat=VCFv4.1									
##samtoolsVersion=0.1.18 (r982:295)									
##INFO=<ID=DP,Number=1,Type=Integer,Description="Raw read depth">									
##INFO=<ID=DP4,Number=4,Type=Integer,Description="# high-quality ref-forward bases, ref-reverse, alt-forward and alt-reverse bases">									
##INFO=<ID=MQ,Number=1,Type=Integer,Description="Root-mean-square mapping quality of covering reads">									
##INFO=<ID=FQ,Number=1,Type=Float,Description="Phred probability of all samples being the same">									
##INFO=<ID=AF1,Number=1,Type=Float,Description="Max-likelihood estimate of the first ALT allele frequency (assuming HWE)">									
##INFO=<ID=AC1,Number=1,Type=Float,Description="Max-likelihood estimate of the first ALT allele count (no HWE assumption)">									
##INFO=<ID=G3,Number=3,Type=Float,Description="ML estimate of genotype frequencies">									
##INFO=<ID=HWE,Number=1,Type=Float,Description="Chi^2 based HWE test P-value based on G3">									
##INFO=<ID=CLR,Number=1,Type=Integer,Description="Log ratio of genotype likelihoods with and without the constraint">									
##INFO=<ID=UGT,Number=1,Type=String,Description="The most probable unconstrained genotype configuration in the trio">									
##INFO=<ID=CGT,Number=1,Type=String,Description="The most probable constrained genotype configuration in the trio">									
##INFO=<ID=PV4,Number=4,Type=Float,Description="P-values for strand bias, baseQ bias, mapQ bias and tail distance bias">									
##INFO=<ID=INDEL,Number=0,Type=Flag,Description="Indicates that the variant is an INDEL.">									
##INFO=<ID=PC2,Number=2,Type=Integer,Description="Phred probability of the nonRef allele frequency in group1 samples being larger (,smaller) than in group2.">									
##INFO=<ID=PCHI2,Number=1,Type=Float,Description="Posterior weighted chi^2 P-value for testing the association between group1 and group2 samples.">									
##INFO=<ID=QCHI2,Number=1,Type=Integer,Description="Phred scaled PCHI2.">									
##INFO=<ID=PR,Number=1,Type=Integer,Description="# permutations yielding a smaller PCHI2.">									
##INFO=<ID=VDB,Number=1,Type=Float,Description="Variant Distance Bias">									
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">									
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">									
##FORMAT=<ID=GL,Number=3,Type=Float,Description="Likelihoods for RR,RA,AA genotypes (R=ref,A=alt)">									
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="# high-quality bases">									
##FORMAT=<ID=SP,Number=1,Type=Integer,Description="Phred-scaled strand bias P-value">									
##FORMAT=<ID=PL,Number=G,Type=Integer,Description="List of Phred-scaled genotype likelihoods">									
#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	mapmerge/AF11-104.bam
scaffold10001_3442	1037	.	C	T	52.3	.	DP=5;VDB=0.0012;AF1=1;AC1=170;DP4=0,0,0,5;MQ=50;FQ=-23.2	GT:PL:DP:SP:GQ	0/1:0,0,0:0:0:3
scaffold10003_3436	3013	.	C	T	63.5	.	DP=16;VDB=0.0377;AF1=0.2441;AC1=41;DP4=13,0,3,0;MQ=50;FQ=64.7;PV4=1,0.3,1,0.055	GT:PL:DP:SP:GQ	0/1:39,3,0:1:0:7
scaffold10003_3436	3095	.	T	C	4.31	.	DP=16;VDB=0.0062;AF1=0.1329;AC1=22;DP4=13,1,1,1;MQ=50;FQ=4.71;PV4=0.24,0.001,1,1	GT:PL:DP:SP:GQ	0/1:44,6,0:2:0:9
scaffold10003_3436	3326	.	TAAAAA	TAAAAAA	134	.	INDEL;DP=12;VDB=0.0030;AF1=1;AC1=170;DP4=0,0,0,12;MQ=50;FQ=-32.2	GT:PL:DP:SP:GQ	1/1:0,0,0:0:0:6
scaffold10004_3428	2366	.	C	G	70.3	.	DP=15;VDB=0.0000;AF1=0.3335;AC1=57;DP4=8,1,6,0;MQ=50;FQ=72;PV4=1,0.22,1,0.22	GT:PL:DP:SP:GQ	0/0:0,0,0:0:0:4
scaffold10004_3428	2427	.	GAAAAAAA	GAAAAAAAA,GAAAAAAAAA	18.5	.	INDEL;DP=17;VDB=0.0054;AF1=1;AC1=170;DP4=0,0,15,0;MQ=50;FQ=-30.8	GT:PL:DP:SP:GQ	0/1:0,0,0,0,0,0,0:0:0:3
scaffold10007_872	485	.	A	G	74.2	.	DP=3;VDB=0.0105;AF1=1;AC1=170;DP4=0,0,3,0;MQ=50;FQ=-23.5	GT:PL:DP:SP:GQ	1/1:0,0,0:0:0:3
scaffold10007_872	747	.	G	A	24.8	.	DP=2;VDB=0.0141;AF1=1;AC1=170;DP4=0,0,0,2;MQ=50;FQ=-22.8	GT:PL:DP:SP:GQ	0/1:0,0,0:0:0:3
scaffold10008_3408	210	.	T	G	29.8	.	DP=2;VDB=0.0125;AF1=1;AC1=170;DP4=0,0,2,0;MQ=50;FQ=-22.8	GT:PL:DP:SP:GQ	0/1:0,0,0:0:0:3
scaffold10008_3408	211	.	T	A	30.8	.	DP=2;VDB=0.0125;AF1=1;AC1=170;DP4=0,0,2,0;MQ=50;FQ=-22.8	GT:PL:DP:SP:GQ	0/1:0,0,0:0:0:3

Figure B.1 – Example VCF file. For the sake of brevity, only 10 rows are displayed. The first 26 rows are meta-information represented as key-value pairs, followed by the header with 8 mandatory columns. The FORMAT column indicates the presence of genotype data. For the sake of brevity, the data of only one individual is shown (*mapmerge/AF11-104.bam*).

Progeny

Individual name	Gender	Patch	Patch.Commune	Patch.Network	Patch.Village	Population	Pedigree name
PT11-101-4	F					PT	PT11-101
PT11-102-5	F					PT	PT11-102
PT11-103-5	F					PT	PT11-103
PT11-106-1	F					PT	PT11-106
PT11-107-3	F					PT	PT11-107
AF11-104	F	1093	Sund	4	Bomarsund	AL	AF11-1093-11
AF11-8757	F	1093	Sund	4	Bomarsund	AL	AF11-1093-8
AF11-5308	M	1183	Lemland	1	Vessingsboda	AL	AF11-1183-2
AF11-16376	F	119	Saltvik	45	Nääs	AL	AF11-119-3
AF11-5171	M	1189	Lemland	1	Vessingsboda	AL	AF11-1198-11

Figure B.2 – Example Progeny file. For the sake of brevity, only 10 rows are shown. Example illustrates how individuals are sourced from two different populations: PT (Pikku Tytärsaari) and AL (Åland Islands). For AL population, habitat patch information including patch and network numbers, patch commune and village are provided. Additionally gender information is provided (F=female, M=male).

Appendix C – SSIS Artefacts

VCFExport.dtsx control flow task

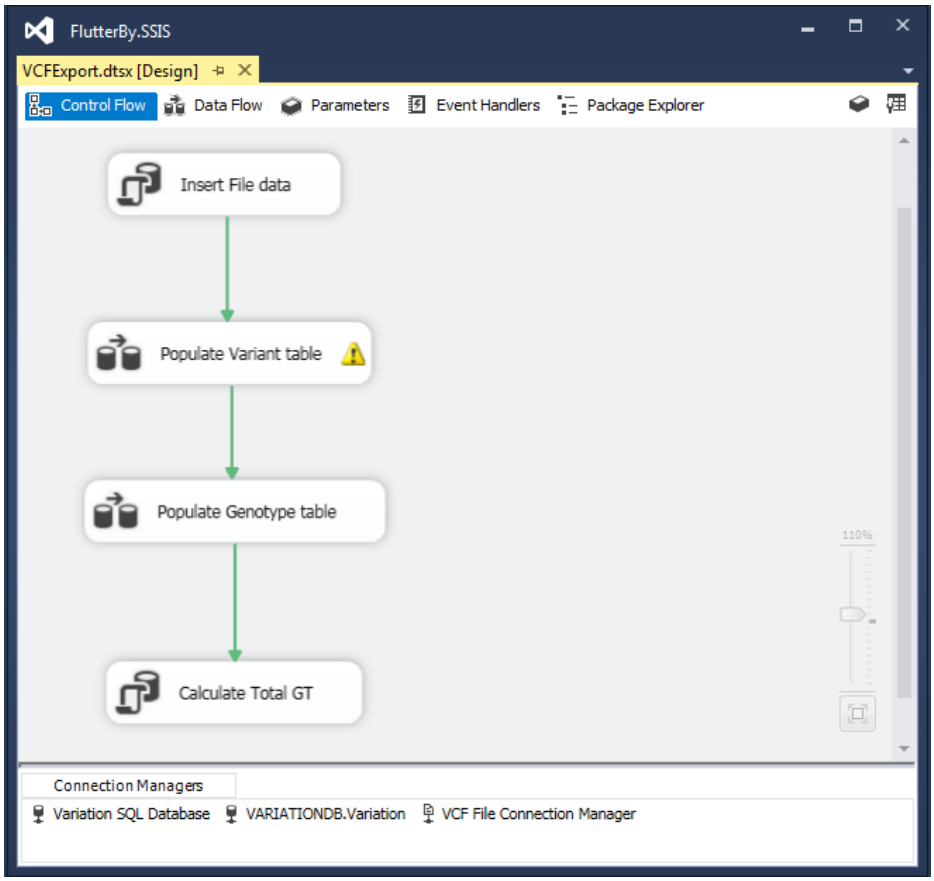


Figure C.1 - Control Flow Designer for the *VCFExport* SSIS package. The developer can switch between Control Flow and Data Flow from the top toolbar pane. The canvas area describes the workflow. The "Connection Managers" tab at the bottom of the canvas lists the available connections in the package.

VCFExport.dtsx "Populate Variant Table" data flow task

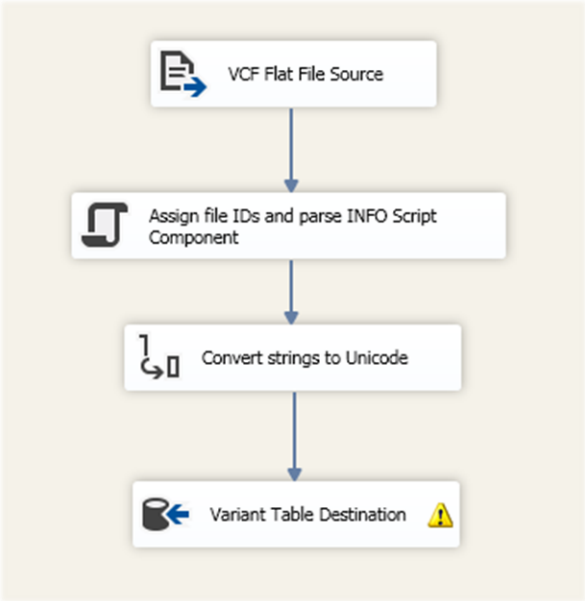


Figure C.2 - Data flow designer for the *Populate Variant Table* data flow task. This illustrates the drill-down view of the *Populate Variant table* control flow task illustrated in Figure C.1.

VCFExport.dtsx "Populate Genotype Table" data flow task

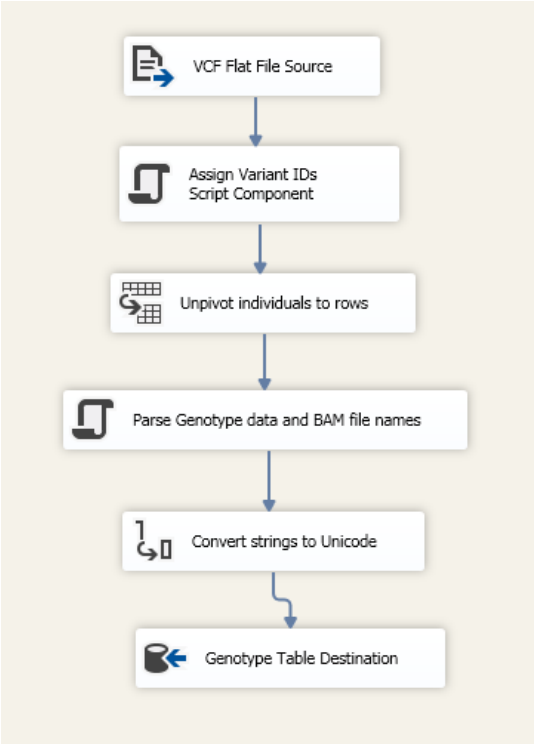


Figure C.3 - Data flow designer for the *Populate Genotype Table* data flow task. This illustrates the drill-down view of the *Populate Genotype table* control flow task illustrated in Figure C.1.

"Calculate Total GT" execute SQL task syntax

```
CREATE TABLE #tmpGT (VariantId INT, GT NVARCHAR(50))
GO
INSERT INTO #tmpGT (VariantId, GT)
SELECT
                        dbo.Genotype.VariantId,
                        dbo.CalculateTotalGT(dbo.Genotype.VariantId) AS GT
FROM            dbo.Variant INNER JOIN
                        dbo.Genotype ON dbo.Variant.Id = dbo.Genotype.VariantId
                                        GROUP BY VariantId
UPDATE Genotype
SET Genotype.CalculatedGT = G1.GT
FROM (SELECT VariantId, GT FROM #tmpGT) AS g1
WHERE Genotype.VariantId = g1.VariantId
AND Genotype.GQ >= 20
AND Genotype.DP > 0
GO
```

Figure C.4 *Calculate Total GT* SQL task from the *VCFExport* SSIS package

fn_CalculateTotalGT user-defined function

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      ER
-- Create date: 18th August 2014
-- Description: Returns the total GT in the format (Ref/Ref, Ref/Alt, Alt/Alt).
-- =============================================
CREATE FUNCTION CalculateTotalGT(@VariantId INT)

RETURNS NVARCHAR(50)
AS
BEGIN
      -- Declare the return variable here
      DECLARE @TotalGT AS VARCHAR(50);
      DECLARE @GT1 AS INT;
      DECLARE @GT2 AS INT;
      DECLARE @GT3 AS INT;

      -- Add the T-SQL statements to compute the return value here
      SELECT @GT1 = COUNT(GT)
            FROM dbo.Genotype AS G1
            WHERE VariantId = @VariantId
            AND G1.DP > 0
            AND G1.GQ >= 20
            AND Gt = '1/1'


      SELECT @GT2 = COUNT(GT)
            FROM dbo.Genotype AS G2
            WHERE VariantId = @VariantId
            AND G2.DP > 0
            AND G2.GQ >= 20
            AND Gt = '0/1'
```

```
	SELECT @GT3 = COUNT(GT)
		FROM dbo.Genotype AS G3
		WHERE VariantId = @VariantId
		AND G3.DP > 0
		AND G3.GQ >= 20
		AND Gt = '0/0'

	-- Return the result of the function
	SET @TotalGT = CAST(@GT1 AS VARCHAR(50)) + '/' + CAST(@GT2 AS VARCHAR(50)) +
'/' + CAST(@GT3 AS VARCHAR(50))

	RETURN @TotalGT

END
GO
```

Figure C.5 – *fn_CalculateTotalGT* user-defined function. Called by the *Calculate Total GT* SQL task during package execution.

Appendix D – System Information

This appendix describes the hardware and software of systems used in the development of the project.

Development environment

OS	Windows 7
Processor	Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz, 2301 MHz, 4 Core(s), 8 Logical Processor(s)
Installed Physical Memory (RAM)	8,00 GB
MongoDB C# Driver	1.9.2
SQL Server Version	SQL Server 2014

MongoDB server

OS	Microsoft Windows Server 2012 R2
Processor	2x AMD Opteron(tm) Processor 4171 HE, 2095 MHz, 4 Core(s), 4 Logical Processor(s)
Installed Physical Memory (RAM)	14,00 GB
MongoDB Version	2.4.8

SQL server

OS	Microsoft Windows Server 2012 R2
Processor	Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz, 2200 MHz, 2 Core(s), 2 Logical Processor(s)
Installed Physical Memory (RAM)	14,00 GB
SQL Server Version	SQL Server 2014

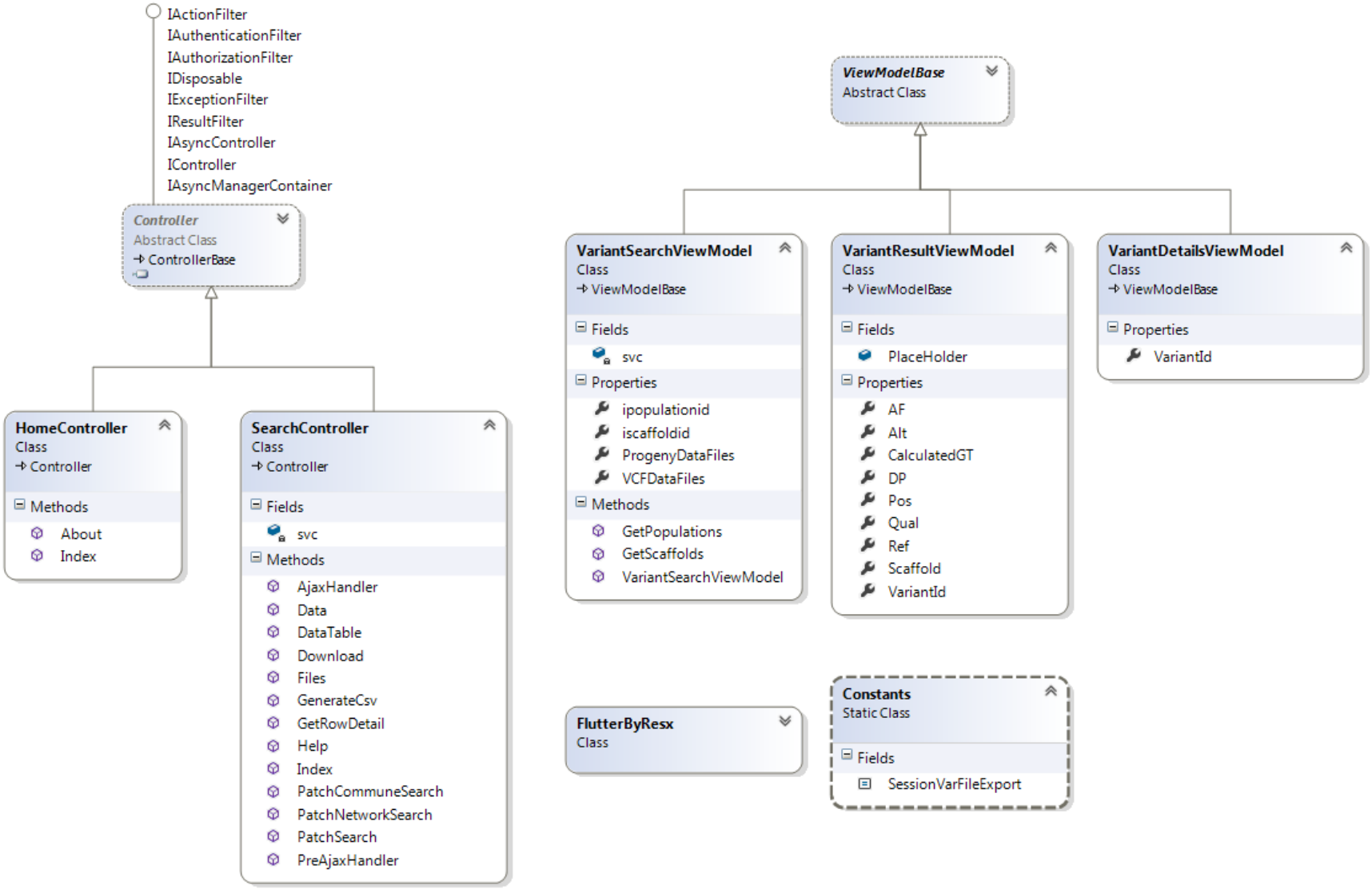
Appendix E – Class Diagrams

Class diagrams

Class diagrams provide a visual representation of the structure of the solution using a notation known as *UML* (Unified Modelling Language). The intention is to provide a high-level overview of how the application was built. The class diagrams illustrated below (Figures E.1, E.2) are based on the projects described in the *Solution Structure* section.

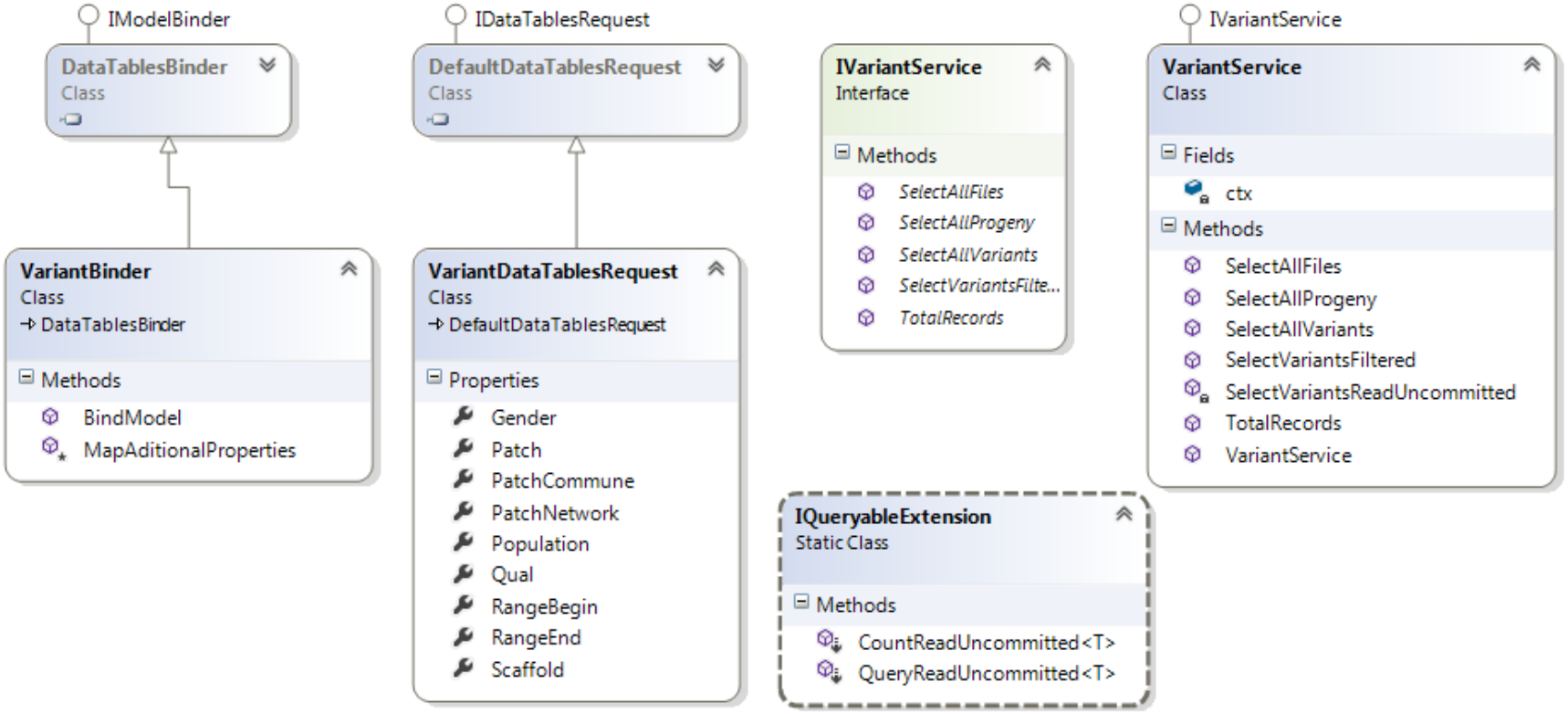
Main project class diagram

Figure E.1 – Code-named *FlutterBy*, the main project class diagram (shown on next page). On the left hand side, the classes that inherit from the *Controller* base class are illustrated. The *Home* controller represents actions for displaying the home page. The *SearchController* represents all of the actions required by the *Search* view. On the right hand side, the classes that inherit from *ViewModelBase* are illustrated. *ViewModelBase* is an empty, abstract class used to define the relationship between other view models. The **ViewModel* classes are used to map data to properties between the controller and the user interface. *VariantSearchViewModel* represents the constructs required for binding the search form with prepopulated data. The *VariantResultViewModel* represents the properties required for displaying search results. The *VariantDetailsViewModel* represents search results that are shown in the data grid master-detail view. *FlutterByResx* is used to store static strings in a single location. This allows common text to be modified from a single location and additionally will allow the application to be translated to different languages in the future. *Constants* is a static class that stores constant strings i.e. static string values that cannot change during the lifetime of the application.


IActionFilter
IAuthenticationFilter
IAuthorizationFilter
IDisposable
IExceptionFilter
IResultFilter
IAsyncController
IController
IAsyncManagerContainer
Controller
Abstract Class
ControllerBase
HomeController
Class
Controller
Methods
About
Index
SearchController
Class
Controller
Fields
svc
Methods
AjaxHandler
Data
DataTable
Download
Files
GenerateCsv
GetRowDetail
Help
Index
PatchCommuneSearch
PatchNetworkSearch
PatchSearch
PreAjaxHandler
ViewModelBase
Abstract Class
VariantSearchViewModel
Class
ViewModelBase
Fields
svc
Properties
ipopulationid
iscaffoldid
ProgenyDataFiles
VCFDataFiles
Methods
GetPopulations
GetScaffolds
VariantSearchViewModel
VariantResultViewModel
Class
ViewModelBase
Fields
PlaceHolder
Properties
AF
Alt
CalculatedGT
DP
Pos
Qual
Ref
Scaffold
VariantId
VariantDetailsViewModel
Class
ViewModelBase
Properties
VariantId
FlutterByResx
Class
Constants
Static Class
Fields
SessionVarFileExport

Data project class diagram

Figure E.2 – *FlutterBy.Data* project class diagram. On the left hand side, the *VariantBinder* class inherits from the *DataTableBinder* class that originates from the *DataTables.Mvc* library which provides the custom methods for binding the *VariantDataTablesRequest* model to the controller. This takes care of binding the custom properties to the request when a search form is submitted in order to filter results. On the far right hand side, the *VariantService* class represents the service class for making calls against Entity Framework. As the search functionality is read only, the class only supports the ability to select records. The necessary methods are exposed by the interface *IVariantService*. The *IQueryableExtension* static class provides helper methods for wrapping database queries in a TransactionScope object.



Appendix F – Source Code

Solution structure

Figure F.1 illustrates the solution structure from the 'Solution Explorer' view in Visual Studio.

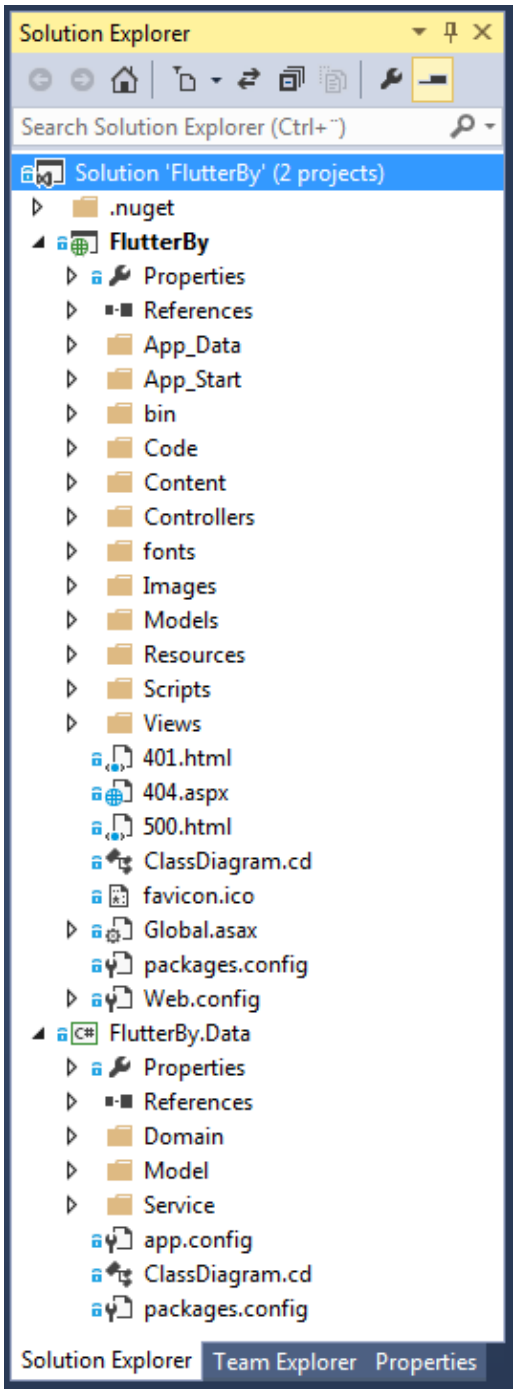


Figure F.1 – Visual Studio solution structure. The solution, code-named "FlutterBy" illustrates a typical solution structure for an ASP.NET MVC web application. The solution is divided into two projects. "FlutterBy" is the main project that holds the web application. The "FlutterBy.Data" project contains the Entity Framework model and the classes responsible for performing data operations.

Code snippets

This appendix shows selected code snippets considered to be of relevance.

VariantService class

The below snippet illustrates the *SelectVariants* method from the *VariantService* class. The *VariantService* class is responsible for acting as a service between the controller and the database context.

```
private IQueryable<Variant> SelectVariantsReadUncommitted(VariantDataTablesRequest
requestModel)
{
  using (var scope = new TransactionScope(TransactionScopeOption.Required, new
TransactionOptions() { IsolationLevel =
System.Transactions.IsolationLevel.ReadUncommitted }))
  {
    IQueryable<Variant> allResults = ctx.Variant.AsNoTracking();
    var query = allResults;

    if (requestModel.RangeBegin != null)
    {
      query = query.Where(v => v.Pos >= requestModel.RangeBegin);
    }

    if (requestModel.RangeEnd != null)
    {
      query = query.Where(v => v.Pos <= requestModel.RangeEnd);
    }

    if (requestModel.Scaffold != null)
    {
      query = query.Where(v => requestModel.Scaffold.AsEnumerable<string>().Any(c =>
c.Contains(v.Chrom)));
    }

    if (requestModel.Qual != null)
      query = query.Where(v => v.Qual >= requestModel.Qual);
    else
      query = query.Where (v => v.Qual >= 20);

    if (requestModel.Population != null || requestModel.Gender != null ||
requestModel.Patch != null || requestModel.PatchCommune != null ||
requestModel.PatchNetwork.HasValue)
    {
      if (requestModel.Population != null)
      {
        query = (from v in query
              join g in ctx.Genotype.AsNoTracking() on new { Id = v.Id } equals new
{ Id = g.VariantId }
              join p in ctx.Progeny.AsNoTracking() on new { BamFile = g.BamFile }
equals new { BamFile = p.IndividualName }
              where (requestModel.Gender == null ||
p.Gender.Equals(requestModel.Gender, StringComparison.OrdinalIgnoreCase))
```

```
                && requestModel.Population.Any(pop => pop.Substring(0,
2).Contains(p.Population))
                && (requestModel.Patch == null || requestModel.Patch.Equals(p.Patch))
                && (requestModel.PatchCommune == null ||
requestModel.PatchCommune.Equals(p.PatchCommune))
                && (!requestModel.PatchNetwork.HasValue || (p.PatchNetwork.HasValue ?
requestModel.PatchNetwork == p.PatchNetwork : (!p.PatchNetwork.HasValue)))
                select v);
      }
      else
      {
        query = (from v in query
                join g in ctx.Genotype.AsNoTracking() on new { Id = v.Id } equals new
{ Id = g.VariantId }
                join p in ctx.Progeny.AsNoTracking() on new { BamFile = g.BamFile }
equals new { BamFile = p.IndividualName }
                where (requestModel.Gender == null ||
p.Gender.Equals(requestModel.Gender, StringComparison.OrdinalIgnoreCase))
                && (requestModel.Patch == null || requestModel.Patch.Equals(p.Patch))
                && (requestModel.PatchCommune == null ||
requestModel.PatchCommune.Equals(p.PatchCommune))
                && (!requestModel.PatchNetwork.HasValue || (p.PatchNetwork.HasValue ?
requestModel.PatchNetwork == p.PatchNetwork : (!p.PatchNetwork.HasValue)))
                select v);
      }
    }
    query = query.Distinct().OrderBy(v => v.Pos).AsQueryable<Variant>();

    var sortedColumn = requestModel.Columns.GetSortedColumns().FirstOrDefault();

    if (sortedColumn != null)
    {
      switch (sortedColumn.Name)
      {
        case "Scaffold":
          if (sortedColumn.SortDirection == Column.OrderDirection.Ascendant)
            query = query.OrderBy(v => v.Chrom).AsQueryable<Variant>();
          else
            query = query.OrderByDescending(v => v.Chrom).AsQueryable<Variant>();
          break;
        case "Pos":
          if (sortedColumn.SortDirection == Column.OrderDirection.Ascendant)
            query = query.OrderBy(v => v.Pos).AsQueryable<Variant>();
          else
            query = query.OrderByDescending(v => v.Pos).AsQueryable<Variant>();
          break;
        case "Refseq":
          if (sortedColumn.SortDirection == Column.OrderDirection.Ascendant)
            query = query.OrderBy(v => v.Ref).AsQueryable<Variant>();
          else
            query = query.OrderByDescending(v => v.Ref).AsQueryable<Variant>();
          break;
        case "Altseq":
          if (sortedColumn.SortDirection == Column.OrderDirection.Ascendant)
            query = query.OrderBy(v => v.Ref).AsQueryable<Variant>();
          else
            query = query.OrderByDescending(v => v.Ref).AsQueryable<Variant>();
          break;
```

```
        default:
          // Default action - filter by pos
          query = query.OrderBy(v => v.Pos).AsQueryable<Variant>();
          break;
      }
    }
    scope.Complete();
    return query;
  }
}
```

The method return type *IQueryable* is a type of Linq query operator supporting deferred execution for querying against remote data sources (e.g. databases). In other words, this method returns a queryable collection of type *Variant* where the query object has been shaped by input parameters. The beginning of the method signature `IQueryable<Variant>` specifies that the IQueryable object queries objects of type *Variant* where *Variant* is an Entity Framework model class containing the methods and properties that describe a variant. The method accepts a parameter of type `VariantDataTablesRequest requestModel` (inherited from DataTables.Mvc *DefaultDataTablesRequest*) that services binding DataTables to controllers to provide additional properties to build the query parameters.

The method syntax is wrapped in a *TransactionScope* object (for affecting database operations as a single transaction) (Microsoft, 2014m). A local variable is created from the database context in which the query is built based on checking the conditions of input parameters (from the `requestModel` parameter). The query is then filtered by distinct records and input parameter is checked against custom sorting settings and if they are present (this depends if the user has clicked a sort button in the UI), the results are sorted by the given sorting criteria and the query is returned to the caller.

The service class in which the method belongs to is instantiated in the controller by creating an object of type `IVariantService`. The `SelectVariantsFiltered` method which in turn calls the private method `SelectVariantsReadUncommitted` is called via the `AjaxHandler` and `GenerateCsv` action methods which are involved in displaying results to the grid and exporting results to CSV.

The `AjaxHandler` and `GenerateCsv` actions will be further described in this appendix.

AjaxHandler action method

The `AjaxHandler` action method belongs to the *SearchController* class. `AjaxHandler` is responsible for returning results to the DataTable which is invoked on the successful submission of the search

form. DataTables constructs a data object that contains the search parameters sent as an AJAX request to the server. The derived implementation of the *DataTables.Mvc* library handles the binding of the DataTables object to the action method without having to directly interrogate HTTP request parameters.

The total records are queried via the `IVariantService` variable already in scope required in order to construct the paging. The filtered results are retrieved by passing the bound `VariantDataTablesRequest` object to the service and the results are partitioned by the paging requirements (the starting position and the number of results per page). The results are cast to an in-memory object (*IEnumerable*) of string arrays in order to be displayed by the DataTable. The results are returned as JSON as DataTables provides built-in support for JSON data sources obtained via AJAX. The DataTables.Mvc library also handles building the appropriate response object (`DataTablesResponse`) consisting of the results and properties required to construct pagination.

```
[ValidateAntiForgeryToken]
public ActionResult AjaxHandler([ModelBinder(typeof(VariantBinder))]
VariantDataTablesRequest requestModel)
{
  var totalRecords = svc.TotalRecords();
  var filteredResults = svc.SelectVariantsFiltered(requestModel);
  var displayedResults =
filteredResults.Skip(requestModel.Start).Take(requestModel.Length);

  //Results need to be IEnumerable in order to be realised by DataTables
  var result = (from v in displayedResults.AsEnumerable<Variant>()
          select new string[]
          {
            string.Empty /*this empty element is a dummy placeholder for the
expander column*/,
            v.Id.ToString(),
            v.Chrom,
            v.Pos.ToString(),
            v.Ref,
            v.Alt,
            v.AF.ToString(),
            v.DP.ToString(),
            v.Qual.ToString(),
            v.Genotype.Where(t => !string.IsNullOrEmpty(t.CalculatedGT)).Select(t =>
t.CalculatedGT).FirstOrDefault() ?? "N/A"
          });

     return Json(new DataTablesResponse(requestModel.Draw, result,
filteredResults.Count(), totalRecords));
}
```

Note: Exception handling is removed for the sake of brevity.

DataTable initialisation

This section lists the JavaScript embedded in the *index.cshtml* view in order to initialise the DataTable. The *onSuccess* callback displays the DataTable when the search form is successfully submitted:

```
var onSuccess = function (result) {
  $("#grid-results").show();

  var exportdt = false;
  var dtOptions = initializeDataTable(exportdt)

  try {
    dataTable = $('table#variantDataTable').dataTable(dtOptions);
  }
  catch(err) {
    alert('An unhandled error occurred. Detailed error for Administrator: ' + err);
    $("#grid-results").hide();
  }
};
```

The `initializeDataTable` function returns configured DataTable options:

```
function initializeDataTable() {
  dtOptions = ({
    paging: true,
    serverSide: true,
    ajax: {
      url: "@Url.Action("AjaxHandler", "Search")",
      type: "POST",
      data: function (d) {
        d.__RequestVerificationToken = $('#SearchForm
input[name=__RequestVerificationToken]').val();
        if ($("#scaffold").val()) {
          d.Scaffold = $("#scaffold").select().val().toString();
        };
        d.RangeBegin = $("#range-begin").select().val().toString();
        d.RangeEnd = $("#range-end").select().val().toString();
        d.Qual = $("#qual").select().val().toString();
        if ($("#gender").find(":selected").index() != 0) {
          d.Gender = $("#gender").select().val().toString();
        };
        if ($("#population").val()) {
          d.Population = $("#population").select().val().toString();
        };
        d.Patch = $("#patch").select().val().toString();
        d.PatchCommune = $("#patch-commune").select().val().toString();
        d.PatchNetwork = $("#patch-network").select().val().toString();
      },
      error: handleAjaxErrorLoc
    },
    language: {
      processing: "<img src='/Images/indicator.white.gif'> Loading..."
    },
    processing: true,
```

```
    destroy: true,
    ordering: true,
    lengthChange: true,
    filter: false,
    order: [],
    columns: [
      {
        name: "Expand",
        orderable: false,
        render: function (data, type, row) {
          return '<div class="foundicon-plus detail" style="color:#000000;
cursor:pointer"></div>';
        }
      },
      {
        name: "VariantId",
        visible: false
      },
      { name: "Scaffold" },
      { name: "Pos" },
      { name: "Refseq" },
      { name: "Altseq" },
      { name: "AF", orderable: false },
      { name: "DP", orderable: false },
      { name: "Qual", orderable: false },
      { name: "GT", orderable: false }
    ],
  });
  return dtOptions;
}
```

GenerateCsv action method

The `GenerateCsv` action method belongs to the *SearchController* class. `GenerateCsv` is responsible for generating the CSV file based on the same bound parameters already submitted to display the DataTable. Similarly to the `AjaxHandler` method, results are retrieved from the service except in this case the results are not partitioned as it is expected that all results that fit the criteria will be exported to file. An instance of the `CsvFileDescription` class is created from the *LinqToCsv* library which describes settings for working with `CsvContext` objects. The file name is generated based on the session ID and stored in a session variable. A new instance of `CsvContext` is created which is used to write the results of the query to the CSV file. The download of the file is handled by the *success* event from the calling AJAX function. Therefore, the `GenerateCsv` does not return any data directly and the calling AJAX function directs control to the `Download` action result after successful completion of generating the CSV for download.

```
[ValidateAntiForgeryToken]
public ActionResult GenerateCsv([ModelBinder(typeof(VariantBinder))]
VariantDataTablesRequest requestModel)
{
```

```
    var filteredResults = svc.SelectVariantsFiltered(requestModel);

    CsvFileDescription outputFileDescription = new CsvFileDescription
    {
      SeparatorChar = '\t', // tab delimited
      FirstLineHasColumnNames = true, // no column names in first record
      FileCultureName = "en-US",
      EnforceCsvColumnAttribute = true
    };

    var fileName = this.HttpContext.Session.SessionID;
    Session.Add(Constants.SessionVarFileExport, fileName);

    var path = Path.Combine(Server.MapPath("~/App_Data/Download"), fileName);

    var cc = new CsvContext();

    using (TextWriter writer = new StreamWriter(path))
    {
      //This needs to be IQueryable, we don't wish to materialize the data set until
it is written to file
      var query = (from v in filteredResults.AsQueryable<Variant>()
              select new VariantResultViewModel()
              {
                VariantId = v.Id,
                Scaffold = v.Chrom,
                Pos = v.Pos,
                Ref = v.Ref,
                Alt = v.Alt,
                AF = v.AF,
                DP = v.DP,
                Qual = v.Qual,
                CalculatedGT = v.Genotype.Where(t =>
!string.IsNullOrEmpty(t.CalculatedGT)).Select(t => t.CalculatedGT).FirstOrDefault()
              });


      cc.Write(query, writer, outputFileDescription);
    }
  return Content("");
}
```

The AJAX call from *index.cshtml*:

```
$.ajax(
  {
    url: "@Url.Action("GenerateCsv", "Search")",
    type: "POST",
    data: settings.oAjaxData,
    success: function (data) {
      document.location.href = "@Url.Action("Download", "Search")";
    },
  });
```

Note: Exception handling is removed for the sake of brevity.

Appendix G – Database

Conceptual database model

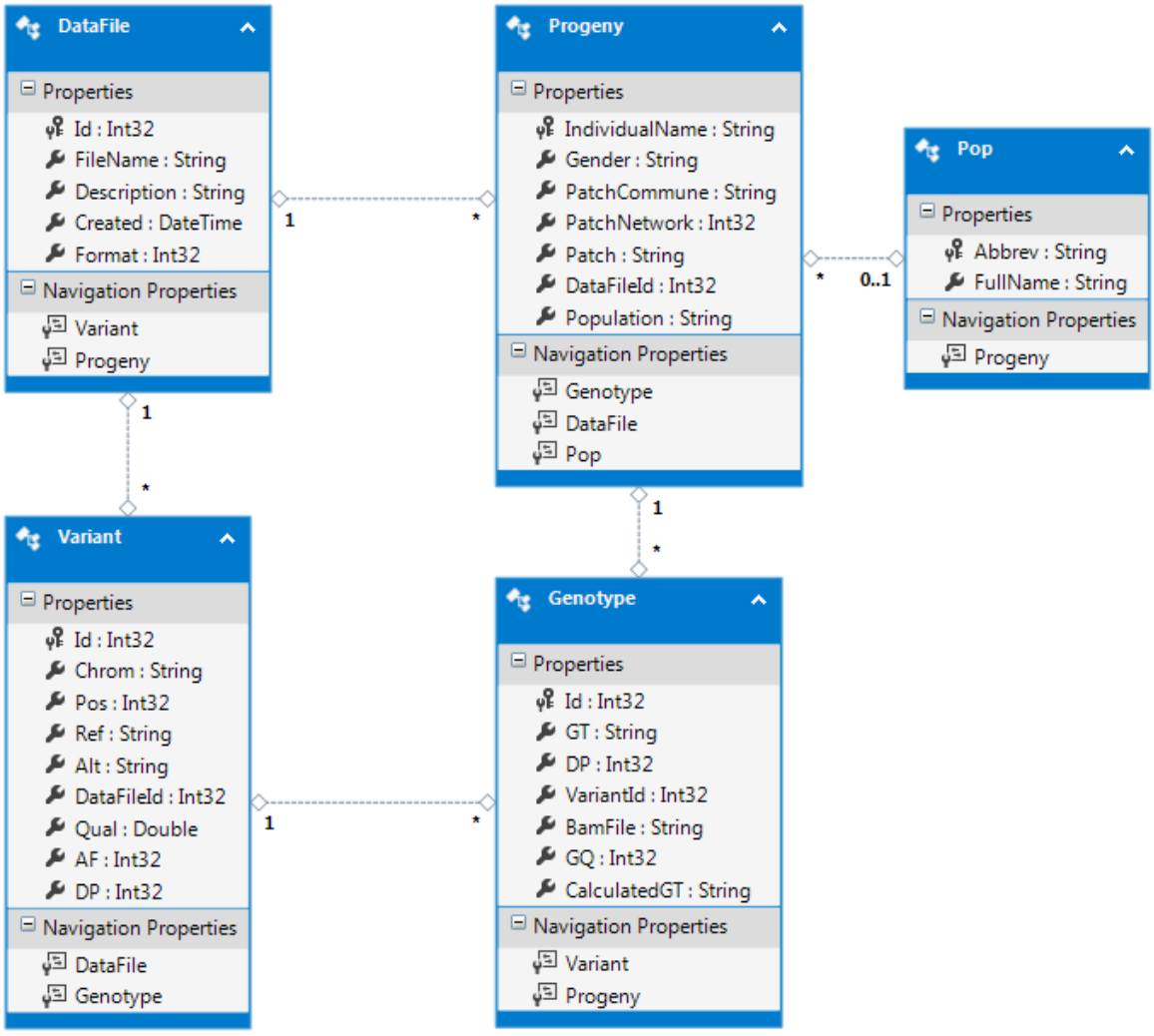


Figure G.1 – Conceptual database model. Entity Designer view in Visual Studio depicting the project conceptual database model. The schema is normalised to reduce redundancy.

DataFile table

The *DataFile* table stores information about data files in order to provide filtering capabilities based on the *FileName* attribute. Additionally, extra information about the file can be stored in the *Description* field, and the created date (*Created*) is automatically populated when a record is added. The *Variant* and *Progeny* tables are related by the *DataFile* primary key field (Id).

Variant table

The *Variant* table contains variant records imported from VCF. The INFO field in VCF has been parsed into *AF* and *DP* columns. The remaining columns (apart from the primary and foreign keys) are mapped directly from VCF (*Chrom, Pos, Ref, Alt,* and *Qual*).

Genotype table

The *Genotype* table contains genotype information from VCF. The *BamFile* column represents the individual genotype field. As a variant from the *Variation* table can contain many individuals, this is modelled as a one-to-many relationship.

Progeny table

The *Progeny* table contains records extracted from the Progeny data file. The *Progeny* table is related to the *Genotype* table via a foreign key constraint between the *IndividualName* and *BamFile* columns.

Pop table

The *Pop* table is a simple look-up table to provide the full names of populations from their two-letter abbreviations.