UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

BIOINFORMATICS APPLICATION FOR ANALYSIS AND VISUALISATION OF ALTERNATIVE SPLICING IN CANCER

Nuno Daniel Saraiva Agostinho

MESTRADO EM INFORMÁTICA

Trabalho de projeto orientado por:
Prof. Doutor André Osório e Cruz de Azerêdo Falcão
e por Dr. Nuno Luís Barbosa Morais

2016

Acknowledgments

There always comes a moment in life where one is asked to accomplish an impossible task. To me, that task is to thank all the people who have always been by my side, as I find that words alone cannot describe my appreciation for you. Nonetheless, I can only hope you understand how grateful I am to all of you.

Starting with Lina, Marie, Teresa, Mariana, Carolina and Juan, you were the best colleagues anyone could ever ask for. Thanks for making this last year pass in a blink of an eye. I learned a lot from you and I had a great time. I would also like to thank Ana Rita, Margarida and Ana Duarte for their friendship and all their help.

Thank you, Nuno. You are a great mentor and an even greater friend. I appreciate all your effort in teaching me about a myriad of exciting fields (and for being the first person that made me understand statistics). Additionally, thanks for helping me improve my oral and written communication to be ever so clear and precise.

André immensely helped me with all the suggestions and support. You and Prof. Antónia are of the greatest teachers I have ever had and it is thanks to people like you that I am motivated to give my best every day.

For being always there to both annoy me and laugh with me, I thank you, Catarina (Inês). It is always a pleasure to constantly annoy you and I find it rather annoying to hear your brilliant suggestions (even though only 5% of your suggestions are worth listening to). Anyway, keep going on with your brilliant suggestions.

João, there is not much that I could say about you that you would not already know about. You are uplifting but nauseating, friendly but tiresome, helpful but maddening. You always know how to make me smile and how to tick me off. Thanks for enduring my lack of patience and also for exploiting it for your own entertainment. You are the best.

Miguel, Zé and Alicia: although you are not as present in my life as I would like, you were always there for me when I needed the most. Thank you all.

Finally, I would like to thank my parents, my brothers and my remaining family for being my greatest source of love, support and headaches. Specially, to my grandfather: science has been part of my life ever since I appeared into yours. I hope you'll stay by my side for a long time.

I couldn't be here without any of you. And if I could go back, I'd go through this all over again and again. And once more. Thank you all.

A quem partilhou o seu tempo comigo, obrigado.

Resumo

A evolução das tecnologias de processamento de dados tem permitido avanços significativos na área das ciências da vida. As melhorias na quantidade e na qualidade dos dados disponíveis para análises biológicas têm proporcionado o estudo em larga escala de diversos processos biológicos.

O *splicing* alternativo é um mecanismo molecular que contribui significativamente para a criação de proteínas com funções distintas a partir do mesmo gene. Encontrando-se este processo envolvido no controlo de muitos mecanismos celulares, a sua desregulação pode promover a progressão de um vasto leque de doenças. Por exemplo, existem associações descritas entre alterações de *splicing* alternativo e a maioria das características de cancro (como evasão ao sistema imunitário e crescimento celular auto-sustentado). Estas associações podem ser analisadas através de dados disponíveis online, como a partir dos dados do The Cancer Genome Atlas (TCGA) que incluem dados clínicos e do perfil molecular de pacientes humanos com diversos tipos de tumores.

A análise e interpretação correcta dos resultados requer competências multi-disciplinares em biologia, estatística e informática. Como nem todos os cientistas das ciências da vida se sentem confortáveis a utilizar ferramentas com uma interface baseada em linha de comandos, vários programas com uma interface gráfica têm emergido para auxiliar na quantificação, análise e visualização de dados biológicos. As análises também podem ser facilitadas através da utilização de dados previamente processados que estão disponíveis publicamente para casos comuns de utilização, evitando-se o processamento dispendioso de dados a nível de tempo e aliviando também a carga na largura de banda relativamente à transferência de dados brutos.

Infelizmente, as ferramentas existentes para a análise de *splicing* alternativo focam-se primariamente na sua quantificação ou apresentam funcionalidades limitadas para a análise subsequente dos eventos de *splicing* alternativo. Além disso, muitos programas que quantificam o *splicing* alternativo utilizam dados brutos e não tiram partido dos dados processados disponíveis de fontes públicas como os dados do TCGA. Assim sendo, existe a necessidade de criar um programa interactivo e fácil de usar que se dedique à análise subsequente dos dados para auxiliar tanto na exploração como no estudo diferencial de dados do *splicing* alternativo, permitindo assim a potencial descrição de novos mecanismos envolvidos na progressão de doenças. Ademais, a integração da informação clínica associada (ausente na maioria dos programas disponíveis) poderá ajudar na identificação de factores de prognóstico e de alvos terapêuticos.

Todos os membros do laboratório de Biologia Computacional do Instituto de Medicina Molecular (Faculdade de Medicina da Universidade de Lisboa) são partes interessadas (*stakeholders*) no projecto, já que apoiam o desenvolvimento do programa e serão utilizadores finais deste. O grupo efectua diariamente muitas das análises e visualizações incorporadas no projecto, tendo ajudado no seu desenvolvimento ao examinar os detalhes de cada análise.

Para o desenvolvimento de uma ferramenta útil, os requisitos necessários foram comunicados pelas partes interessadas ao longo de várias reuniões. Acordou-se que a aplicação se deve focar em obter dados a partir de fontes *online* (como do TCGA), processar, carregar e manipular os dados na aplicação, quantificar *splicing* alternativo e analisar estatisticamente os dados disponíveis (por exemplo, análise de sobrevivência, componentes principais e diferencial de *splicing*), incluindo funcionalidades para criar e editar grupos baseados nos dados clínicos e para gravar os resultados obtidos (conforme apropriado). Outras características de interesse incluem a capacidade de adicionar novos repositórios, reconhecer novos formatos aquando da identificação e do carregamento de ficheiros, acrescentar novas ferramentas para manipulação de dados e incorporar novas análises e visualizações a partir dos dados carregados.

Consoante os requisitos funcionais discutidos, os atributos não funcionais incluem a capacidade de modificação (fácil de modificar e introduzir novas componentes de sistema), a usabilidade (interface fácil de usar e consistente que mostre mensagens de erro e de aviso informativas), o desempenho (foco no tempo tomado pelas operações dado a quantidade de dados para processar e analisar) e a capacidade de reposta (informar o utilizador da operação a decorrer através de uma barra de progresso e bloqueando o botão de início de uma acção durante a operação).

Dada a importância da análise estatística e biológica no projecto e do interesse da comunidade científica no R e no Bioconductor (repositório de pacotes R associados a dados biológicos), foi decidido que o projecto seria desenvolvido com base no Shiny, uma *framework* para desenvolvimento de aplicações *web* que permite construir aplicações interactivas com a linguagem R e incorporar gráficos interactivos desenvolvidos em HTML5 e JavaScript. Todas as funcionalidades destas ferramentas foram testadas e estudadas através de um protótipo antes da concepção da arquitectura.

A arquitectura do sistema foi desenhada de forma modular e extensível, consoante os requisitos mencionados, estimulando assim contribuições de quaisquer partes interessadas, bem como facilitando a expansão do seu suporte para outras fontes de dados, formatos de ficheiro e análises e visualizações efectuadas sem necessidade de alterar as funcionalidades básicas. Para além de facilitar testes e correcções às unidades do programa, a expansibilidade possibilita actualizar as ferramentas com novos métodos explorados e desenvolvidos na área e, consequentemente, aumentar o interesse da comunidade científica no programa.

A modularidade foi implementada de forma a que, quando o utilizador chama a função para começar a interface visual do programa, dá-se início a uma série de chamadas hierárquicas a outras funções do programa, as quais preparam a interface e a lógica de todos os módulos disponíveis. O programa é composto pelos módulos de obtenção de dados (para obter dados locais ou do TCGA e processá-los de acordo com o seu formato), quantificação de *splicing* alternativo, análise de dados (análises clínica, componentes principais e diferencial e informações associadas aos genes dos eventos de *splicing*), agrupamento de dados e definições do programa. Os gráficos interactivos foram adaptados ao diferentes módulos conforme apropriado, recorrendo ao pacote Highcharter.

A aplicação realiza vários testes automáticos para validar o *output* das unidades do programa, de forma a alertar o programador caso haja alguma mudança que altere o *output* esperado. Esta funcionalidade está incorporada em ferramentas de testes contínuos como o Travis-CI e o AppVeyor. A cobertura do código testado também é avaliada pela ferramenta CodeCov.

Para testar a interface do programa, foram realizados testes de usabilidade a 6 membros do grupo

de Biologia Computacional do Instituto de Medicina Molecular no seu ambiente de trabalho, consoante diversas tarefas pré-definidas. Os participantes escolhidos representam a público-alvo: utilizadores proficientes no conhecimento do domínio de interesse. Várias métricas foram medidas durante as sessões de teste, incluindo o número de problemas encontrados e as opiniões dos utilizadores sobre cada tarefa. Cada participante utilizou uma versão que tentou melhorar alguns dos problemas encontrados pelo participante anterior. A interface foi considerada, em média, muito boa ou excelente para cada tarefa e permitiu atentar a 45 problemas distintos (actualmente, pelo menos 25 desses problemas já foram resolvidos). Para melhorar a usabilidade e funcionalidade do programa, solicitámos recentemente o parecer de contribuidores externos especializados na análise de *splicing* alternativo.

A aplicação também foi testada a nível de desempenho para vários tipos de tumores. Uma análise completa para dados associados a pacientes com cancro da mama (cerca de 1097 pacientes, o maior número de pacientes disponíveis para qualquer tipo de tumor no TCGA) demora cerca de 6 minutos através da interface visual: 47 segundos para carregar os dados necessários do TCGA (excluindo tempo de transferência), 2 minutos e 39 segundos para quantificar o *splicing* alternativo e 2 minutos e 35 segundos para a análise diferencial baseada nas amostras normais versus tumorais. A interface visual adiciona um *overhead* ao desempenho, daí ter sido escolhida para medir os tempos nos piores casos possíveis.

Em suma, temos estado a desenvolver uma aplicação *web* em R com uma interface gráfica para a quantificação, análise integrada e visualização de dados de *splicing* alternativo a partir de grandes conjuntos de dados transcriptómicos provenientes do projecto The Cancer Genome Atlas (TCGA). Esta ferramenta interactiva realiza análise de componentes principais e outras análises exploratórias graficamente assistidas. Entre os seus aspectos mais inovadores encontra-se a análise de variância (que a pesquisa do grupo revela como importante na detecção de alvos de interesse de outra forma despercebidos) e a incorporação de dados clínicos (como estádio tumoral e dados de sobrevivência) associados com as amostras do TCGA. De interesse também se encontra incorporado o acesso visual interactivo para mapeamento genómico e anotação funcional dos eventos de *splicing* alternativo seleccionados. Aplicação desenvolvida permitiu revelar assinaturas de *splicing* alternativo específicos de cancro e novos factores putativos de prognóstico.

O código da ferramenta já se encontra gratuitamente disponível através de uma licença MIT no GitHub (`http://github.com/nuno-agostinho/psichomics`) e foi enviada para ser aceite no Bioconductor. Actualmente, a aplicação apenas pode ser utilizada localmente, mas há planos para a disponibilizar num servidor *web* do Instituto de Medicina Molecular.

Palavras-chave: Bioinformática, cancro, *splicing* alternativo, visualização

Abstract

Alternative splicing (AS) allows proteins with distinct functions to be generated from the same gene, being involved in the control of many common cellular processes. Its deregulation may therefore foster the progression of a wide range of diseases. For instance, associations between most of the hallmarks of cancer and AS alterations have been reported.

The advent of next-generation sequencing has allowed the profiling of transcriptomes beyond gene expression, enabling genome-wide studies of AS. However, the currently available tools for the analysis of AS from RNA-Seq data are not user-friendly and primarily focus on quantification, having limited features for downstream analysis.

To overcome these limitations, we have been developing an R application with a graphical interface for the integrated analysis of AS from large transcriptomic datasets, namely from The Cancer Genome Atlas (TCGA) project. The tool interactively performs clustering, principal component and other graphically-assisted exploratory analyses. Amongst its innovative aspects are the analysis of variance (which our research shows to be important in the detection of otherwise unnoticed putative targets) and the direct incorporation of clinical features (such as tumour stage or survival) associated with TCGA samples. Interactive visual access to genomic mapping and functional annotation of selected AS events is also incorporated. We have successfully used the application in the revelation of cancer-specific AS signatures and associated novel putative prognostic factors.

The application's architecture is modular and extensible, aiming to stimulate contributions from its users, as well as to gradually expand its support to other data sources and file formats and the scope of its analysis and visualisation tools without modifying its core functionalities. The tool is available in GitHub (`http://github.com/nuno-agostinho/psichomics`) and was submitted to be accepted in Bioconductor. Currently, the application is locally run but there are plans to deploy it in a web server from Instituto de Medicina Molecular.

Keywords: Bioinformatics, cancer, alternative splicing, analysis, visualisation

Contents

List of Figures

List of Tables

Acronyms

API Application Program Interface. 2, 43, 54

CI Continuous Integration. 17, 47, 51

CLI Command-Line Interface. 1

CRAN The Comprehensive R Archive Network. xix, 19, 20, 34, 35, 47, *Glossary:* The Comprehensive R Archive Network

CSS Cascading Style Sheets. 12, 19, 23, 39, 40

DNA deoxyribonucleic acid. xix, 5, 7, 8, *Glossary:* deoxyribonucleic acid

GTEx Genotype-Tissue Expression project. 9, 54

GTF Gene Transfer Format. 13

HTML HyperText Markup Language. 19, 23, 34, 37, 38, 40, 51

IDE Integrated Development Environment. 11, 20

JSON JavaScript Object Notation. 31, 37

mRNA messenger RNA. xix, 5–8, 12, 13, 16, 32, 37, *Glossary:* messenger RNA

PCA principal component analysis. 15, 35

PSI percent spliced-in. xix, 7, 8, 14, 16, 17, 36, 45, 54, 63, 64, *Glossary:* percent spliced-in

REST Representational State Transfer. 11, 12, 31, 38, 39

RNA ribonucleic acid. xix, 5, 7–9, *Glossary:* ribonucleic acid

RNA-seq RNA sequencing. xix, 7, 9, 31, *Glossary:* RNA sequencing

TCGA The Cancer Genome Atlas. xix, 1, 2, 9, 12, 14, 16, 23, 25, 31, 32, 34, 50, 53, 54, 67, *Glossary:* The Cancer Genome Atlas

TSV Tab-separated Values. 32, 37

TXT text file. 13

UCSC University of California Santa Cruz. 13, 32, 38

UTR untranslated region. xx, 5, 6, *Glossary:* untranslated region

VCS Version Control System. 17

XML Extensible Markup Language. 12

Glossary

Bioconductor Repository of R packages dedicated to biological data analyses. 12, 17, 19, 20, 47, 51, 53

deoxyribonucleic acid (DNA) Molecule that stores the genetic information of a cell. xix, 5

gene DNA sequence that is transcribed to RNA. 5–7, 9, 12, 13, 17, 34, 37, 38, 45

Highcharter R package to create JavaScript-based interactive plots. 2, 11, 12, 35–37, 39

messenger RNA (mRNA) RNA molecule used as a template for protein synthesis. xix, 5

percent spliced-in (PSI) Metric used to calculate the proportion of reads that support the inclusion isoform. xix, 7

read Sequencing reads are short text strings corresponding to a sequence of DNA or RNA. 7–9, 12–15, 25, 31

ribonucleic acid (RNA) Molecules transcribed from genes and that are involved in protein synthesis, gene regulation, DNA replication and other cellular functions. xix, 5

RNA sequencing (RNAseq) Sequencing method to detect and quantify RNAs from a sample. xix, 7

Shiny R package to create interactive web applications using a reactive programming model. 2, 12, 19, 20, 23, 26, 29–31, 34, 38, 39, 42, 43, 47

The Cancer Genome Atlas (TCGA) Project that aims to catalogue data (including clinical information and molecular profiles) from samples of human patients regarding diverse tumour types.. xix, 1

The Comprehensive R Archive Network (CRAN) Repository of generic R packages. xix, 19

untranslated region (UTR) Ends of the mRNA sequence not translated into protein. UTRs may regulate the translation mechanism of its mRNA. xx, 5

Chapter 1

Introduction

The fast development of data processing technologies has had a large impact on life sciences research. The ever-increasing amount and resolution of available data for biological analyses has recently allowed the study of a vast variety of biological processes in a large scale [1].

One of the molecular mechanisms that most significantly contributes to protein diversity is alternative splicing [2,3]. Given the regulatory roles of this process in cellular metabolism, its deregulation is linked with disease development, including cancer [4–8] and neurodegeneration [8,9]. This association can be analysed by retrieving biological data from online sources, such as The Cancer Genome Atlas (TCGA), a project that catalogues clinical and molecular profiling data of different kind of human tumours [10].

However, correct analyses and interpretation of results require a multi-disciplinary expertise in biology, statistics and computer science. As not every life scientist is comfortable using tools based on a Command-Line Interface (CLI), many programs with a visual interface have emerged to assist in the quantification, analysis and visualisation of biological data [11–13]. Processed data is also publicly offered for the most common research uses to facilitate data analyses, overcoming the time-consuming step of data processing and bandwidth concerns related with raw data transference.

Unfortunately, current tools used for studying alternative splicing either present over-simplistic analyses or focus mostly on its quantification [9,14–16]. Furthermore, many of the programs that quantify alternative splicing use raw data without leveraging the available higher-level data from sources such as TCGA [9,14,15].

Therefore, there is a need for an user-friendly and interactive program that goes beyond quantification-only analysis to assist in both exploration and differential study of alternative splicing data, thus allowing the potential characterisation of novel mechanisms involved in disease progression. Moreover, the integration of clinical-related information (which is absent in most available programs) may help in identifying relevant prognostic factors and therapeutic targets.

To overcome these limitations, we have been developing a web application called *PSIchomics* to quantify, analyse and visualise alternative splicing data. This app is an R package with a modular design to easily introduce new features, including the support of new file formats and new data analyses. The app is currently available in GitHub at `http://github.com/nuno-agostinho/psichomics` and it will be released in Bioconductor, a repository of R packages for biological data [17].

1.1 Requirements

All the members of the Computational Biology laboratory at Instituto de Medicina Molecular (Faculdade de Medicina, Universidade de Lisboa) are stakeholders in the project since they have an interest in the program development and they are part of the end-users. They also perform many of the analyses and visualisations of this project daily and aided in the program development by scrutinising the details of each analysis.

The requirements of the project were stated in natural language by the stakeholders through recurrent discussions to minimise any missteps. My background in biology helped in understanding the functional requirements and how to develop the application. Expectedly, not all requirements were stated in the first meetings and some of the requirements evolved with time, reinforcing the need for successive conversations with the stakeholders.

The following was agreed upon: the application must allow data retrieval from online sources (such as retrieval of TCGA data), alternative splicing quantification and statistical data analyses (including survival, principal component and differential splicing analyses) with interactive plots, as appropriate. The diverse analyses also must support the creation and manipulation of data grouping. Also of interest, the program must have an easy-to-use interface and to easily allow to add new data sources, recognise new file formats when loading data, add new manipulation tools and incorporate new analyses and visualisations.

There were no additional constraints, which allowed to freely choose the programming language and the type of program (whether a desktop or web application, a library, etc).

1.2 Contributions

The design, implementation, testing and deployment of the application were conducted by me with suggestions and guidance from my laboratory colleagues and advisors. The duration of the project activities is plotted in figure 1.1.

All the members of the Computational Biology laboratory at Instituto de Medicina Molecular (Faculdade de Medicina da Universidade de Lisboa) are stakeholders in the project since they are part of the future end-users. They also assisted program development, namely the fine details of each data analysis step, given that they commonly perform the type of analyses and visualisations this project focuses on. Additionally, they participated in the usability tests of the program.

Prof. Dr. Antónia Lopes (Faculdade de Ciências da Universidade de Lisboa) also made valuable suggestions to the architecture of the application.

During the application development, I have also contributed to some open-source projects, such as the R package Highcharter with the code of some graphical plots (survival curves, density plots and smaller additions) that are already part of the latest package release [18]. I have also contributed to Shiny with code for a text area input [19]. Additionally, I suggested improving code reusability to promote information hiding in *Firebrowser*, an R package that works as a client to the Firehose Application Program Interface (API) and provides both the data available in TCGA and diverse analyses on TCGA datasets [20]. The authors of *Firebrowser* kindly offered me co-authorship of their manuscript titled *Firebrowser: An R Client to the Broad Institute's Firehose Pipeline* for my participation in the project.

Figure 1.1: Gantt chart with the duration of the project's activities. The project started in September 2015 and ended in September 2016.

The final manuscript, which I also reviewed, was submitted to *Database: The Journal of Biological Databases and Curation* and it is currently under review.

Other notable contributions I made include cross-referencing alternative splicing events from four programs that quantify alternative splicing events (MISO [14], VAST-TOOLS [9], rMATS [15] and SUPPA [16]). This is useful to the Computational Biology group in order to compare results between the mentioned tools. Additionally, I performed the alternative splicing quantification of 116 rat RNA-Seq samples [21] using SUPPA and rMATS for an ongoing research project on toxicotrancriptomics in the lab.

Moreover, we will write a research article on the developed program, to be submitted to the Bioinformatics journal. This project was already presented in a poster session at the EMBO Young Scientists' Forum 2016 (Lisbon, Portugal), a meeting for life scientists.

1.3 Workplace

All the work was developed at Instituto de Medicina Molecular (Faculdade de Medicina da Universidade de Lisboa) with the guidance of Dr. Nuno Morais (group leader of the Computational Biology group) and Prof. Dr. André Falcão (supervisor from Faculdade de Ciências da Universidade de Lisboa).

The Instituto de Medicina Molecular is a research entity focused on biomedical research and it offers

diverse facilities to support the everyday work taking place in the institute, such as dedicated services to microscopy imaging, information technology support and animal facilities[1].

The stakeholders were always available for questions and suggestions which allowed the project to progress nicely thanks to their constant feedback. There were also some focused meetings to discuss the project more deeply, including a presentation of the project to other members of the Instituto de Medicina Molecular, which allowed me to get input on the project from third-parties that knew little or nothing at all about the project.

1.4 Document Structure

The remaining sections of this document are organised as follows:

- Chapter 2 - ”Concepts and Related Work” explores the biological concepts and existing solutions for the quantification and analyses of alternative splicing.
- Chapter 3 - ”Materials and Methods” shortly describes the approaches involved in the work performed.
- Chapter 4 - ”Design Decisions” discusses the choice of the programming language, the development of the prototype and the program’s architecture.
- Chapter 5 - ”Implementation” examines the current implementation of the program and its modules. Also, it features case studies related to the program’s functionality.
- Chapter 6 - ”Testing” details tests performed on the program, including installation, usability, unit and continuous testing. It also features software benchmarking.
- Chapter 7 - ”Deployment” delves into software deployment and open-source licenses.
- Chapter 8 - ”Conclusions” states final remarks, the importance of this project and plans for future work.

[1] More information at `https://imm.medicina.ulisboa.pt` (last accessed on 23 August 2016)

Chapter 2

Concepts and Related Work

2.1 Genetic Information Transfer

Cell functions are dependent on the information encoded in segments of deoxyribonucleic acid (DNA) molecules called genes. Each gene is a segment of DNA that is copied to ribonucleic acid (RNA) in a process referred to as transcription. After this process, messenger RNA (mRNA) molecules carry the information as templates for proteins to be synthesised in a process called translation [22, 23].

The mRNA sequence involved in protein synthesis is not identical to its genes. For instance, the ends of the mRNA sequence, known as untranslated regions (UTRs), operate on functions related to the mRNA translation but they do not serve as part of the protein-coding template. mRNA processing also includes a splicing mechanism that removes some segments (called introns) of the primarily transcribed RNA (pre-mRNA) and keeps its remaining segments (exons) as part of the sequence that will encode for a protein (figure 2.1) [3, 23–25].

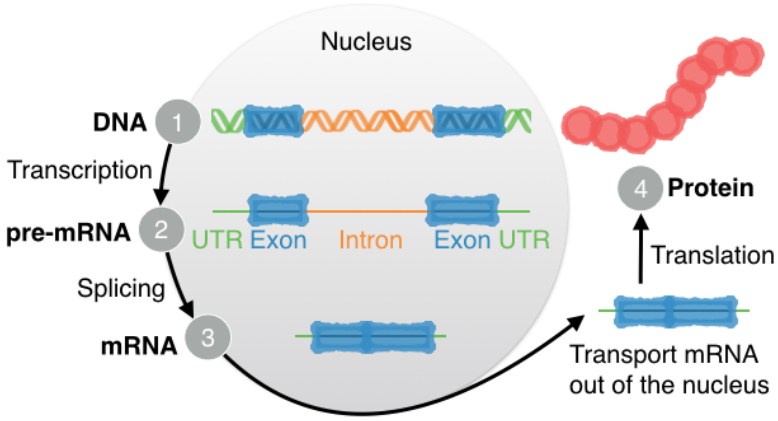


Figure 2.1: pre-mRNA is synthesised from DNA in the cell nucleus and then processed into a mature mRNA (or simply mRNA). Finally, the mRNA is exported out of the nucleus to serve as the template of a new protein.

2.2 Alternative Splicing

The majority of human mRNAs undergo alternative splicing. In this process, the splicing of a single pre-mRNA may occur differently by exon exclusion, exon truncation, intron retention or even a combination of the former that may result in a great diversity of mRNA variants (known as mRNA isoforms) translatable into specific proteins (protein isoforms) [3, 24]. Even though they are encoded by the same gene, protein isoforms may have different or even antagonising functions. One such example is the

KLF6 gene whose protein isoforms are involved in tumour suppressor functions with the exception of one splicing isoform that promotes tumour growth and dissemination [26].

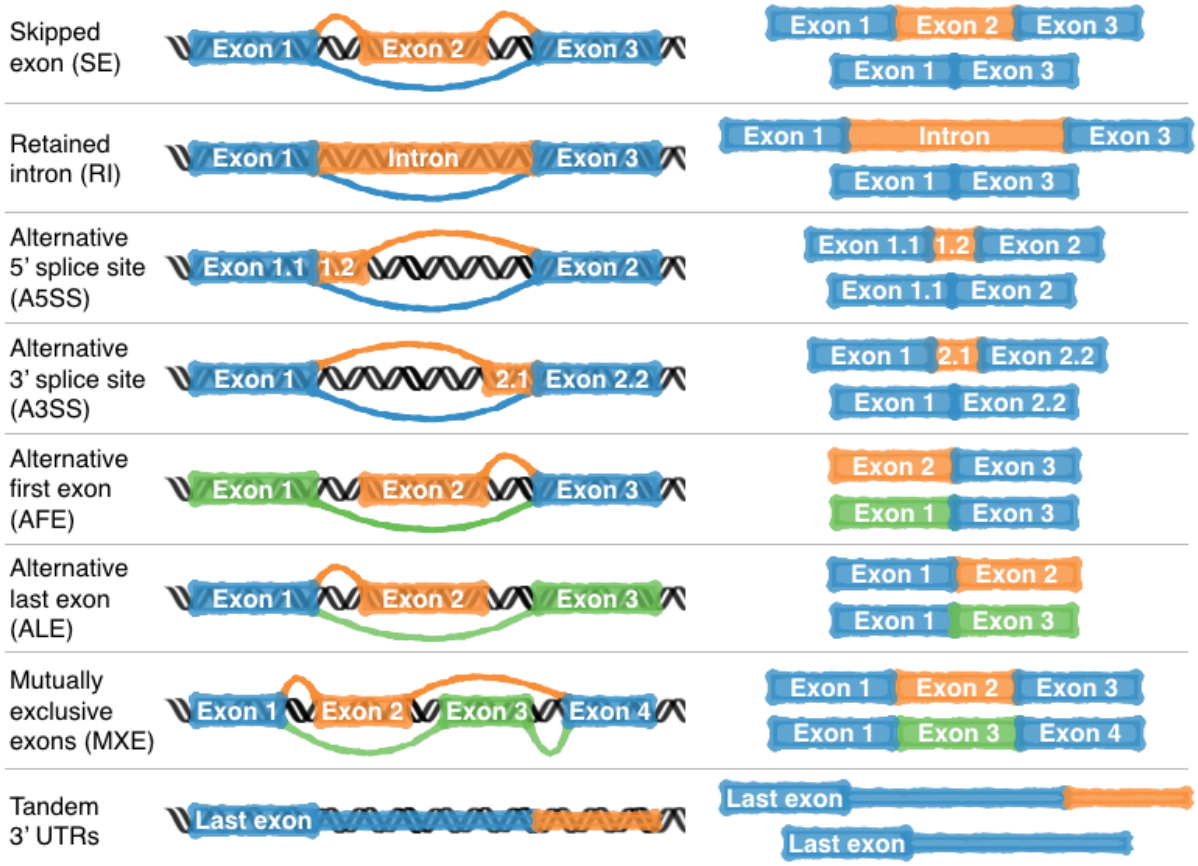


Figure 2.2: Types of alternative splicing events. In the given cases, each gene may express one of the two mRNA variants depicted on the right by inclusion and exclusion of specific exons. Constitutive exons are depicted in blue while alternative ones are depicted in orange and green.

Alternative splicing is common in mammals and invertebrates and particularly abundant in primates. The average human gene has 3,5 isoforms compared to 2,75 in mice, 1,25 in fruit flies and 1,25 in nematodes, which supports an association between alternative splicing and organismal complexity [27, 28]. Interestingly, alternative splicing rates are also tissue-dependent in vertebrates. Alternative splicing is much more frequent in the brain than in other tissues like the heart, liver and kidney [28].

Alternative splicing can occur in different ways which have been categorised in the following types: skipped exon (SE), intron retention (IR), alternative 5' (A5SS) and 3' (A3SS) splice sites, alternative first (AFE) and last (ALE) exon, mutually exclusive exons (MXE) and tandem 3' UTR (depicted in figure 2.2). Although this categorisation is widely used, it is criticised for its inflexibility regarding more complex events [29].

2.2.1 Association with Disease

Alternative splicing is involved in the control of many cellular processes [4]. Expectedly, its deregulation is associated with a wide range of diseases, namely the progress of cancer [4–8] and neurodegenerative disorders [8, 9]. For instance, certain tumour cells inhibit programmed cell death (a defence

mechanism against cancer cells) by skipping exons 3 to 6 in mRNAs from the *caspase 9* gene [30].

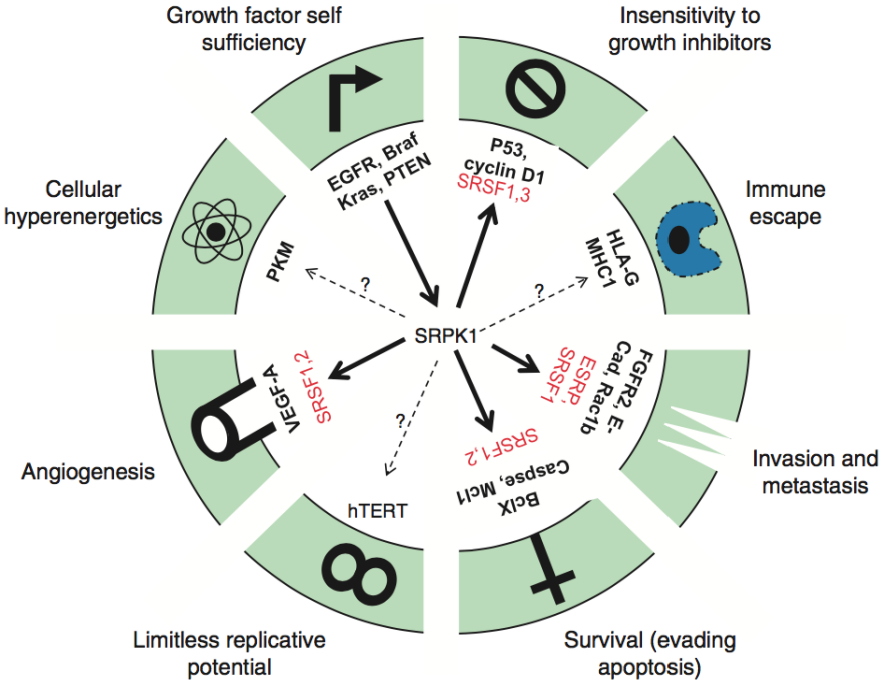


Figure 2.3: Regulators in the hallmarks of cancer are alternatively spliced. The gene SRSF1 is an alternatively spliced regulator that is reported to affect at least four cancer hallmarks. Image retrieved from [4].

Specifically during tumour progression, normal cells may progressively acquire oncogenic properties known as the **hallmarks of cancer**, including cell death evasion, tissue invasion and metastasis, limitless replication, and immune system evasion, among others [4, 31]. These hallmarks have been associated with a diversity of deregulated splicing isoforms [4] as depicted in figure 2.3. Studying this association allows for a better understanding of tumour formation and progression, namely by identifying isoforms involved in cancer development and if they can be potential therapeutic targets.

2.2.2 Quantification

Alternative splicing may be profiled by next-generation RNA sequencing (RNA-seq) [1].This technology yields short RNA sequence text strings (called reads) that are mapped to a DNA of reference [1] (as depicted in figure 2.4) or to a transcriptome (i.e. the set of all RNAs[1] in a cell type or organism) of reference.

The presence of a given exon may be quantified using the percent spliced-in (PSI) metric, corresponding to the proportion of isoforms that include a certain exon [2, 32, 33]. The distribution of PSI values for each alternative splicing event can then be compared between different groups. For instance, distributions with a statistical significance difference between normal and disease samples or between tumour stages may reveal events related to disease progression.

[1]RNAs are also known as transcripts.

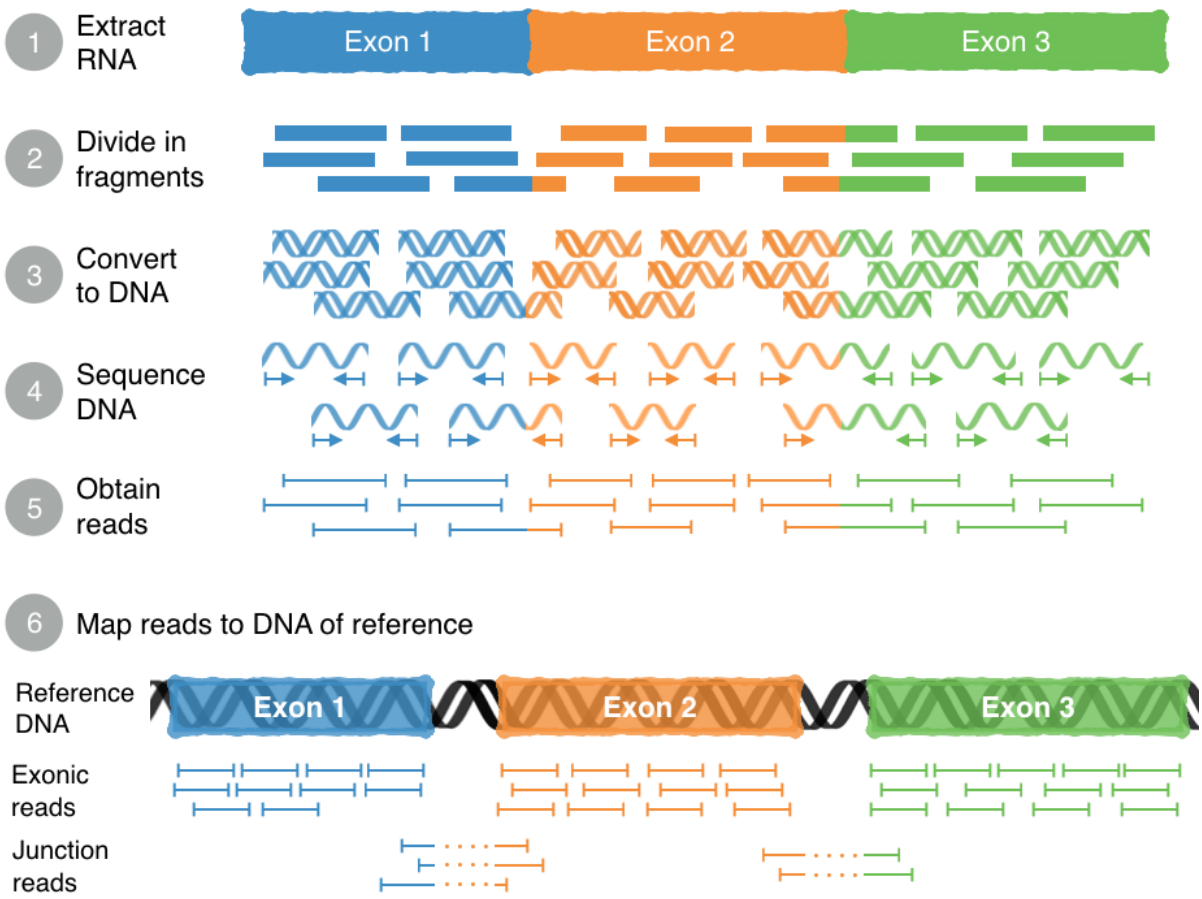


Figure 2.4: RNA sequencing and read mapping. RNA is first extracted from a sample (1) and divided into small fragments (2). Next, these fragments are converted to DNA (3). The DNA fragments are then sequenced, producing short text strings called reads (4). Finally, these reads are mapped to a DNA of reference (5), which allows to reconstruct the extracted mRNAs and identify exon coordinates in the reference DNA.

Quantification Tools

There are several programs that quantify alternative splicing, including MISO [14], AltAnalyze [11], VAST-TOOLS [9], rMATS [15], jSplice [35] and SUPPA [16]. Exon inclusion levels can be calculated using junction reads alone (depicted in the bottom of figure 2.4), as in the case of VAST-TOOLS and jSplice [9, 35], although MISO and rMATS were specifically designed to use additional exon-spanning reads to obtain PSI values with greater precision [14, 15]. Exceptionally, the quantification of intron retention events requires not only junction reads but also mid-intron and retention reads to distinguish these events from other transcript variations (figure 2.5) [34].

The process to quantify alternative splicing events can take a long time if programs use complex and accurate algorithms; for instance, MISO and rMATS use time-consuming Bayesian inference to properly identify to which isoforms the sequencing reads belong to [14,15]. Conversely, programs like SUPPA and jSplice rely on already processed data to estimate those same values, improving the speed of calculations by orders of magnitude while retaining enough accuracy when compared to previous methods [16, 35].

Although the aforementioned tools quantify transcriptomic data, only AltAnalyze and jSplice accept

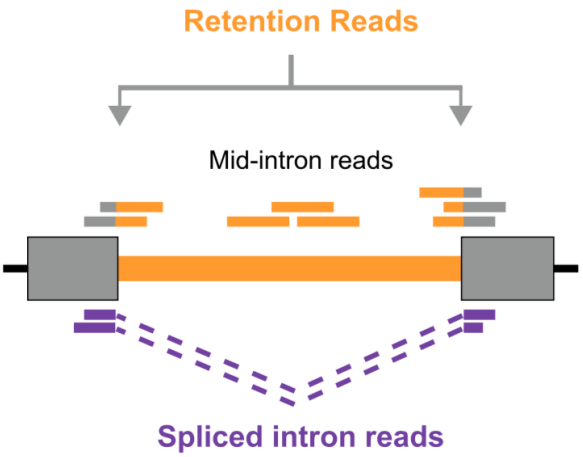


Figure 2.5: Quantification of an intron retention event requires junctions reads (illustrated as spliced intron reads), mid-intron reads and retention (i.e. exon-intron junction) reads. Image retrieved from [34].

junction read counts as input to measure the levels of exon inclusion [11,35]. This allows these programs to quantify data from available large-scale databases like *TCGA* — a human tumour data repository that also includes matched normal samples[2] in a smaller scale [10] — and *Genotype-Tissue Expression project (GTEx)* — a database of multiple normal human tissue data [36]. Usage of both databases is extensively reported in the literature [6, 7, 37, 38].

The quantification of alternative splicing events is followed by statistical analyses, including differential splicing analysis. Although some of the previously mentioned tools also perform downstream analyses of alternative splicing, they mainly focus on its quantification.

2.3 Analytical Tools

There are many tools that analyse transcriptomic data. However, most do not focus on alternative splicing [13,39,40]. The few programs that do, either present over-simplistic differential splicing analysis or no proper downstream analysis to assist its biological interpretation. Some of the current tools for splicing analysis include:

- TIN [41] is an R package to analyse alternative splicing data in cancer; yet, the package does not have a graphical interface; also, instead of using data from RNA-seq, TIN bases the analyses on micro-array[3] data.
- VAST-TOOLS [9] is a command-line tool written in Perl and R that performs limited analyses and inflexible (i.e. not explorable nor personalisable) visualisations from raw data.
- AltAnalyze [11] is a Java program with a graphical user interface that performs limited analyses based on junction reads but its documentation lacks information on splicing quantification.
- SpliceSeq [42], a Java software that identifies alternative splicing patterns in RNA-seq data. A particularly relevant approach is TCGA SpliceSeq [12], a web version of SpliceSeq for TCGA data that can focus on a given gene of interest or on all genes with significant splicing variation and show the respective exon inclusion levels for a given tumour type compared to normal tissue;

[2] Matched normal samples are retrieved from the same tissue where the tumour was found or from blood of the same patient.
[3] An older method to measure gene expression (RNAs) universally used before the advent of RNA-seq with lower resolution.

a particular interesting feature is that it allows to make comparisons across multiple tumour types at once.

- SUPPA [16] is a program developed in Python that got recently updated to calculate differential splicing and cluster alternative splicing events according to their inclusion levels across multiple conditions.

The alternative splicing analyses performable by the listed programs are seriously limited by their inflexibility: they do not allow to dig deeper into the data. For instance, in all cases mentioned, clinical data cannot be used to run survival analysis based on alternative splicing profiles. Therefore, there is a need for a more flexible tool with the capability to process and perform proper downstream analyses of alternative splicing in cancer through the use of an interactive graphical user interface.

Chapter 3

Materials and Methods

3.1 R Statistical Language

The R statistical language (also known as GNU S) is a free cross-platform programming language dedicated to statistical and graphical computation [43]. The R language can be expanded by installing packages available from online repositories.

RStudio [44] is a graphical Integrated Development Environment (IDE) developed to work with R. RStudio Desktop version 0.99.903 with R 3.3.1 was used to develop all the project's code in a machine running OS X 10.11.6 with 4 cores and 8GB of memory. The following packages were used during package development:

- testthat 1.0.2 to create unit tests [45];
- devtools 1.11.1 to facilitate package development [46];
- microbenchmark 1.4.2.1 to accurately measure the execution of R expressions [47];
- profVis 0.3.2 to visualise profiling data from R [48];
- rmarkdown 0.9.6 to create help documents using markdown and R code [49];
- roxygen2 5.0.1 to comment functions [50].

3.1.1 Packages

The following packages are dependencies of the program:

- data.table 1.9.6 to faster subset, update, group and perform set operations on data frames [51];
- digest 0.6.9 to compare MD5 and SHA-1 hash algorithms [52];
- DT 0.2 to render tables with filtering, sorting and searching features, among others [53];
- fastmatch 1.0-4 to reduce look-up times based on hash tables [54];
- Highcharter 0.4.0 to plot R objects using JavaScript [18].
- httr 1.1.0 to retrieve information from services that provide a Representational State Transfer (REST) architectural style [55];
- jsonlite 0.9.21 to parse JSON data [56];

- miscTools 0.6-16 to employ miscellaneous tools and utilities including vectorised functions of interest [57];
- plyr 1.8.4 and dplyr 0.4.3 to manipulate and analyse data [58, 59];
- R.utils to incorporate many programming utilities [60];
- rlist 0.4.6.1 to easily work with lists [61];
- Shiny 0.14 to create a web application [19];
- shinyjs 0.6 to extend the JavaScript operations from Shiny [62];
- Sushi, a Bioconductor package, to visualise genomic data [63];
- XML to parse and generate Extensible Markup Language (XML) files [64].

Standard packages bundled with R used throughout the project bundled with R (like utils, stats and survival) are not listed.

3.1.2 External Libraries

Shiny [19] includes the Javascript libraries *jQuery* 1.12.4, *ion.RangeSlider* 2.1.2 (slider input) and *selectize.js* 0.12.1 (jQuery-based select box). Shiny also includes the *Bootstrap* 3.3.7 (web development framework) and *FontAwesome* 4.6.3 (Cascading Style Sheets (CSS) library for icons). The package Highcharter [18] uses *Highcharts* 4.2.4 (JavaScript-based plots) and the package DT [53] uses *DataTables* 1.10.5 (jQuery-based tables).

The minimised source code of the following JavaScript MIT-licensed libraries are included in the package: *fuzzy.js* 0.1.0 for approximate string matching[1] and *jquery-textcomplete* 1.3.4 to present text completion suggestions from a dropdown menu[2]. The CSS MIT-licensed library *Animate.css* is also included to provide cross-browser animations like fade in and out[3].

3.2 Data Retrieval

Firehose-formatted TCGA data (like clinical information and RNA-seq junction read counts) are retrieved from Firebrowse through its RESTful service [65]. Ensembl [66] and UniProt[4] also provide services based on a REST architecture style which are used to retrieve genetic information like genomic position, mRNAs and proteins of a given gene. Additionally, the PubMed Central's RESTful service is used to retrieve research articles related to the selected alternative splicing event [67].

To retrieve data from the aforementioned RESTful services, the R package httr is used. httr makes it easier to use HTTP methods for RESTful services like GET, POST, PUT, PATCH and DELETE [55].

3.2.1 Alternative Splicing Annotation

An alternative splicing annotation file contains the genomic coordinates of the splice junctions for each splicing event. The annotation file for the Human genome (hg19 assembly) is provided with the

[1] `https://github.com/bripkens/fuzzy.js` (last accessed on 20 September 2016)
[2] `https://github.com/yuku-t/jquery-textcomplete` (last accessed on 20 September 2016)
[3] `https://daneden.github.io/animate.css/` (last accessed on 20 September 2016)
[4] `http://www.uniprot.org/help/programmatic_access` (last accessed on 20 September 2016)

package and was prepared based on the available annotations from the following alternative splicing quantification tools: MISO [14], VAST-TOOLS [9, 34], rMATS [15] and SUPPA [16].

While the alternative splicing event annotation from MISO and VAST-TOOLS can be retrieved online, the annotation from SUPPA and rMATS is only obtainable after running the programs with an mRNA annotation file stating the coordinates of all mRNAs and respective exons. This file is downloadable from the University of California Santa Cruz (UCSC) Table Browser [68] in the Gene Transfer Format (GTF) format as required by both programs. The mRNA annotation file should contain the gene and mRNA identifiers in each line, although the gene identifier from the downloaded file is actually the transcript identifier given a possible bug with the chosen format. To resolve this issue, we retrieved the file in text file (TXT) to create a table with the matches between gene and transcript identifiers, replacing the incorrect gene identifier in the GTF file.

Event annotation files from the different programs were cross-referenced and combined by matching the chromosome, genomic coordinates and strand of the alternative splicing events. This combined annotation is placed alongside the application to be used when estimating alternative splicing quantification. Table 3.1 shows the number of annotated alternative splicing events available from each program.

Table 3.1: Annotated events retrieved from programs that quantify alternative splicing.

Alternative splicing event type		**VAST-TOOLS**	**rMATS**	**SUPPA**	**MISO**
Skipped Exon	SE	142 806	45 983	48 467	47 444
Retained Intron	RI	166 647	5 544	6 774	5 990
Alternative 5' Splice Site	A5SS	13 748	4 906	17 305	12 813
Alternative 3' Splice Site	A3SS	18 007	9 057	16 881	16 665
Mutually Exclusive Exon	MXE		2 333	5 114	2 721
Alternative First Exon	AFE		51 119	75 604	18 989
Alternative Last Exon	ALE		8 958	18 748	9 863
Tandem 3' UTRs				2 656	

3.3 Alternative Splicing Quantification

As previously mentioned (subsection 2.2.2 Quantification), alternative splicing is quantifiable by programs such as MISO [14], AltAnalyze [11], VAST-TOOLS [9, 34], rMATS [15], jSplice [35] and SUPPA [16], but only AltAnalyze and jSplice accept junction read counts as input to measure the levels of exon inclusion [11, 35].

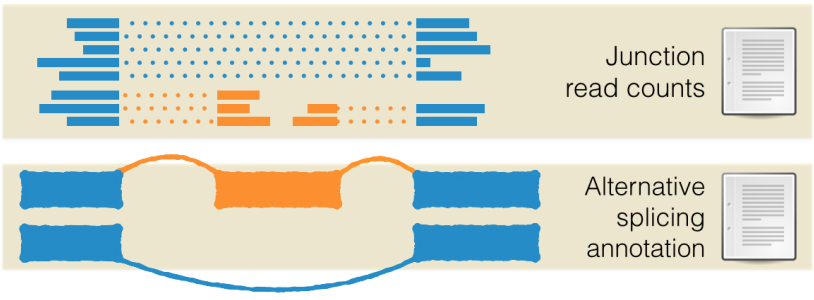


Figure 3.1: Alternative splicing is quantified using (1) an alternative splicing annotation file containing the genomic coordinates of splice junctions and (2) a file with the number of reads aligning with each splice junction (junction read counts) for each sample.

In a similar fashion, our program quantifies alternative splicing from processed data available in online databases, thus skipping the time-consuming step of processing raw data. Alternative splicing quantification is performed using junction quantification from TCGA and the alternative splicing annotation combined from multiple sources (figure 3.1).

The PSI metric is used to quantify alternative splicing through the proportion of isoforms that include a certain exon. This can be estimated by the ratio of normalised number of aligned reads (read counts) that support the inclusion isoform to the normalised reads count supporting the exclusion isoform [2, 32, 33]. The alternative splicing event types supported by the program and the respective formula used to measure PSI values are summarised in table 3.2.

Table 3.2: Quantification of alternative splicing event types using junction read counts supporting the inclusion and exclusion of an exon. C_1A and AC_2 represent read counts supporting junctions between a constitutive and an alternative exon and therefore alternative exon inclusion, while C_1C_2 represents read counts supporting junctions between the two constitutive exons and therefore alternative exon exclusion. The splice junctions are illustrated in figure 3.2 for convenience.

Alternative splicing event type	**Acronym**	**Quantification**
Skipped exon	SE	$\Psi = \frac{(C_1A + AC_2)/2}{(C_1A + AC_2)/2 + C_1C_2}$
Mutually exclusive exon	MXE	$\Psi = \frac{(C_1A_1 + A_1C_2)}{(C_1A_1 + A_1C_2) + (C_1A_2 + A_2C_2)}$
Alternative 5' splice site Alternative first exon	A5SS AFE	$\Psi = \frac{AC_2}{AC_2 + C_1C_2}$
Alternative 3' splice site Alternative last exon	A3SS ALE	$\Psi = \frac{C_1A}{C_1A + C_1C_2}$

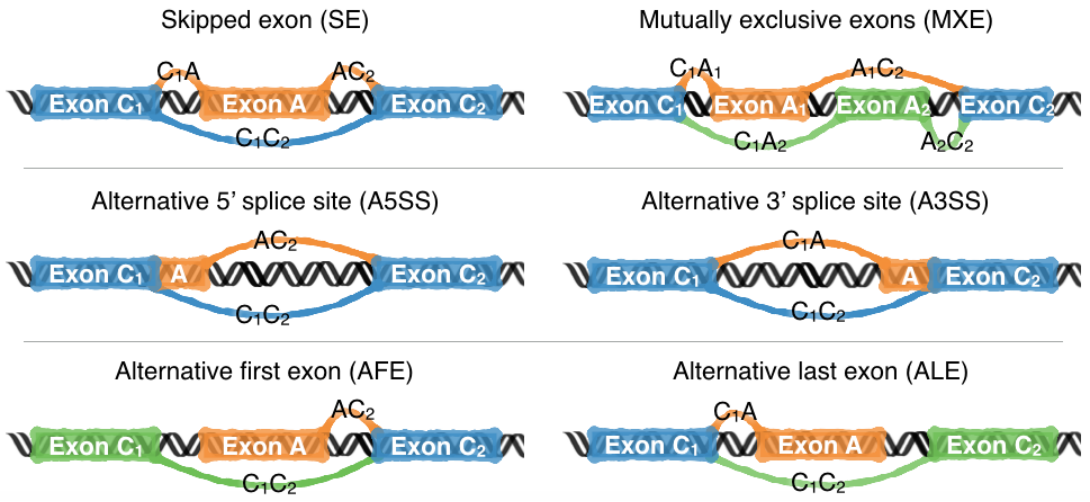


Figure 3.2: Splicing junctions used to measure alternative splicing by event type. Alternative exons are coloured in orange and green, while constitutive exons are coloured in blue. The green alternative exons in alternative first and last exon events are considered as constitutive, as each event can only consist of one alternative exon.

Splicing events with a total read count (i.e. the sum of inclusion- and exclusion-supporting read counts) below a given threshold are discarded from the analyses. By default, this user-adjustable thresh-

old is set to 10 reads.

3.4 Data Analyses

Principal component, survival and differential splicing analyses were employed in the exploration of alternative splicing quantification data and their combination with clinical information, being discussed below.

3.4.1 Principal Component Analysis

Principal component analysis (PCA) is an algorithm to reduce the number of dimensions in a data set by rotating the variables through multiple dimensions and identifying combinations of these variables that characterise most of the variation. These combinations are called principal components and they allow to represent the data with a fewer number of dimensions, thus making it easier to plot and group data samples according to the variance associated to the respective principal components [69].

When performing PCA on a data matrix, no value can be missing. Missing values have to either be removed (implying the removal of a whole row or column) or imputed (replacing them by an operation on remaining values, like the mean or the median). However, the removal of columns for a small number of missing values may have the undesired effect of losing useful information. Instead, a threshold can be defined to ensure that columns with a low number of missing values have them replaced by the median value of the column, while columns with a large amount of missing values are discarded.

3.4.2 Survival Analysis

Survival analysis examines the expected duration of time until the occurrence of one or more events of interest, such as death in the context of clinical data. This kind of analysis can estimate patients' survivability in different conditions (for instance, to compare disease treatments) [70–72]. The tools to perform and plot survival curves are built-in in R through the survival package [73, 74].

Kaplan-Meier curves are commonly used to estimate the proportion of a population that would survive a given length of time under the same circumstances, given a set of observed survival times. To test if two curves are statistically different, the log-rank test is used [70, 72].

According to [72], different types of survival curves are used according to the event of interest:

- **Overall survival curves** use death as the event of interest, which provides a sense of the group survivability;
- **Disease free survival curves** focus on the relapse of a disease as the event of interest and these curves are lower than overall survival curves given that patients may have relapsed but not died;
- **Progression free survival** prioritise the progression of a disease (for instance, tumour growth or spread) as the event of interest to isolate treatment outcomes from the disease;
- **Disease specific survival** or **cause specific survival curves** highlight death from the disease of interest which may be misleading, given the limited events retrieved after removing patients that have disease relapse or non-related deaths (this may even exclude patients that died by treatments or other factors related to the disease).

The clinical data retrieved from TCGA contains useful information for survival analysis including time (in days) to patient's last observation, death, new tumour event after initial treatment and surgical removal, as well as drug and radiation treatment start and end times.

One important aspect of survival analysis is to track when patients drop from the study to distinguish them from the patients that underwent an event of interest. This is relevant to **data censoring**. Events are censored if they either occurred before subject enrolment (left-censoring) or, as it is more usual, if the subject left the study or the study ended before the event occurrence (right-censoring). Censored events are tick-marked in the survival curves [70–72].

Another type of censoring is known as interval-censored survival analysis, which is useful to study events whose exact time of occurrence is unknown, even though data are available before and after the event occurred. However, as none of the stakeholders is sufficiently familiar with this method, it was not a focus of this work.

In order to explore the effects of several variables that affect survival, the proportional hazards regression analysis (also known as Cox regression model or simply Cox model) is used. The Cox model also estimates the risk of death (hazard) for an individual given a set of prognostic variables [70]. Although the Cox model is commonly used in survival analysis, a new statistical approach to analyse the effect of mRNA isoform variation in survival has recently been purposed [75]. This method will be considered for future implementation.

3.4.3 Differential Splicing Analysis

Differential splicing analysis is performed over the quantification of a given alternative splicing event between data groups (for instance, tumour versus normal samples). It usually consists in using statistical tests that make no assumptions about the data distribution. Such statistical tests are known as non-parametric [76–78] and they include:

- The **Wilcoxon rank-sum** or **Mann-Whitney U test** compares the median of two groups and if the observations in one of them tend to be larger than in the other [76, 78];
- The **Wilcoxon signed-rank text** performs an analysis of median ranks to access if two paired groups belong to different populations [78];
- The **Kruskal-Wallis rank sum test** performs a variance analysis to check if two or more groups are similar [76, 78];
- The **Levene's test** tests for variation differences between two or more samples [79].

Although non-parametric tests are valid for most data, they generally have lower statistical power than parametric tests. Finding the correct tools to efficiently test hypotheses from splicing data is challenging. Reportedly, the distribution of exon inclusion levels (PSI values) suggests that each exon tends to be nearly always included or always excluded in a given cell (see figure 3.3) [80]. Therefore, it has been purposed that PSI values follow a beta distribution [14, 81–83]. A beta regression can then be used to model exon inclusion levels directly, using the R package betareg [83–85] that can be integrated in the future.

Given the sheer amount of splicing events profiled, significance must be corrected for multiple testing. The application can apply several p-value adjustment methods from the family-wise error rate (Bonferroni, Holm, Hochberg and Hommel corrections) and false discovery rate (Benjamini-Hochberg and Benjamini-Yekutieli methods) [86].

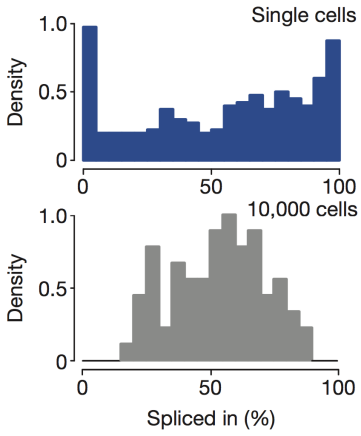


Figure 3.3: PSI distribution of splicing events from highly expressed genes in individual cells (top) and populations (bottom). Adapted from [80].

3.5 Version Control

Version Control Systems (VCSs) allow to track each change made to a working repository and make it easy to check the code history and to revert a bad change [87]. One of the most popular VCSs is **git** for its high-level abstraction compared to older VCSs [88]. The project makes use of git and it is hosted as a public repository in GitHub[5].

RStudio supports common tasks of version control with git, including: (un)stage changes, commit changes, amend last commit, show history log, show file diffs, push and pull, among others [89].

3.6 Testing Tools

3.6.1 Continuous Integration and Code Coverage

After each change to the GitHub repository, the Continuous Integration (CI) software Travis-CI runs `R CMD build` and `R CMD check` on the project[6]. This ensures the package can be built from scratch. Travis-CI tests the package in virtual machines running Ubuntu 12.04.5 LTS Server Edition (64-bit) with 2 cores and 7.5GB of memory[7].

In the end of the automated build, Travis-CI runs CodeCov, a remotely hosted tool that tests a project's code coverage, i.e. what lines of code are being tested or not by the unit tests[8].

The CI service AppVeyor is used to build and test the package on a virtual machine using Windows Server 2012 R2 (64-bit) with 2 cores and 4GB of memory[9].

3.6.2 Benchmarking

To measure the time to load data, quantify alternative splicing (skipped exon) and perform differential splicing analysis, normal versus tumour comparisons were separately run 10 times with the same settings for different tumour types in a machine running OS X 10.11.6 with 4 cores and 8GB of memory. The visual interface of the program was run in Safari 10.0 and RStudio Desktop version 0.99.903 with R 3.3.1.

[5] `https://github.com/nuno-agostinho/psichomics`
[6] These commands need to pass with no errors or warnings for the package to be accepted in Bioconductor (subsection 4.1.1)
[7] `https://docs.travis-ci.com/user/ci-environment/` (last accessed on 14 September 2016)
[8] `https://codecov.io` (last accessed on 14 September 2016)
[9] `https://www.appveyor.com` (last accessed on 14 September 2016)

3.6.3 Usability Testing

Usability testing was performed with 6 members of the Computational Biology Lab at Instituto de Medicina Molecular. The test sessions were conducted according to a detailed script (see section C.1: Script) on a machine running Ubuntu 14.04.4 LTS (64-bit) with 32 cores and 264 GB of memory. The program was run in Firefox 47.0 from RStudio 0.99.902 running R 3.3.1. Each test session lasted between 1.3 to 2 hours.

Chapter 4

Design Decisions

4.1 Programming Language

The free and open-source R programming language focuses on statistical and graphical computation interpreted by a console. The basic functionality of R can be expanded by packages available in online repositories such as The Comprehensive R Archive Network (CRAN) (a general repository of R packages) and Bioconductor (dedicated to store R packages related to biological data analyses [17]).

The R language was chosen over other more commonly-used languages in program development (like Java, C++ or Python) given the interest of the scientific community in R and Bioconductor and the diverse R packages that satisfy many biological data analyses that can be integrated in the package [90–92]. The prospect of biologists contributing with code for custom analysis also influenced this decision. Although there are projects like *rpy2* that run R code embedded in a Python process[1], these community projects have limitations and compatibility problems with some R packages, which may hinder their adoption.

Nevertheless, one of the major limitations of R is the low performance compared to lower-level languages, even though it is possible to mitigate it when using packages like BiocParallel for parallel programming and Rcpp to run C++ code from R [93].

On the other hand, a significant motivation to develop in R is the Shiny package [19], a web application framework that allows to create an entire web app using only R code but also allowing to add custom HyperText Markup Language (HTML), CSS and Javascript for more complex applications. Shiny follows a reactive programming model where the output is bound to the input (reactive dependency) and, every time the input changes, the output automatically updates as well according to the new input [19]. Most R packages can be used within Shiny including JavaScript-based packages that allow for graphical visualisations.

4.1.1 Bioconductor's Package Guidelines

Given that R was selected as the primary programming language, it was also decided to submit the program as an R package to Bioconductor. Bioconductor is an online repository of R packages focused on computational biology and bioinformatics [17] widely used for biological data analysis [92, 94, 95].

[1] `http://rpy.sourceforge.net` (last accessed on 28 August 2016)

The approval of new packages in Bioconductor must follow guidelines[2], including:

- There must be no errors or warnings when building (`R CMD build`), checking the package for problem issues (`R CMD check`) and running Bioconductor-specific package checks (`R CMD BiocCheck`);
- The package size built by `R CMD build` must be less than 4 MB;
- `R CMD check` must complete within 5 minutes;
- The package must contain documentation illustrating the major uses of the package and comprehensive help pages for exported functions;
- The package must not contain code from third-parties that cannot be distributed under the specified license;
- The package must include release notes;
- The package can only depend on R packages available in either CRAN or Bioconductor (a specially limiting requirement).

4.2 Prototype

Before defining the architecture of the application, a prototype was developed to test R, Shiny and the IDE RStudio. Both the prototype and the book [96] allowed to understand more about the R language.

For instance, I learned through them how object-oriented and functional programming works in R. A mixed approach using both paradigms was also tested as [97] suggests but one of its negative aspects is that object-oriented programming in R is much more verbose than in languages like Java and C++. Also, methods are related to functions instead of classes as in traditional object-oriented languages [98].

It is also fundamental to know that, when the user loads an R package, all its functions are loaded as well. However, the source files from the package are not available, only its functions. R also does not support function overloading (functions with the same name but distinct implementation) in the same namespace.

4.2.1 Debugging

R offers specialised debugging functions to fix bugs in the code. One example is `browser()`, which opens a debugging session at an arbitrary code line. This debugging session allows to interactively run arbitrary code from the console and browse the code using commands to execute the next line in a function, step into a called function, finish the execution of the current loop or function and continue regular command execution by exiting the debugging session. RStudio extends the debugging tool by contextually highlighting the debugged line in the source file and showing all defined variables and their corresponding values in the current context [96].

[2] Available at `https://www.bioconductor.org/developers/package-submission/` (last accessed on 12 September 2016)

4.2.2 Code Performance

Performance can be measured using packages like `microbenchmark` to compare multiple runs of different R expressions or functions [47].

To improve code performance in R, it is usual to use vectorised functions which are much faster than running the same code in a for loop. Since the for loops used in vectorised functions are written in C, they have comparatively less overhead. There are many vectorised functions for most common cases in R and for those cases when there is no appropriate vectorised function, it is possible to write one in a language such as C++ using the package Rcpp [93, 96]. Yet another way to improve the performance of a function is to use the byte code compiler integrated. This is integrated in R and allows to compile a function to improve its speed, with execution times comparable to a C version of the same function [96].

There are other practices to improve performance which are more akin to other programming languages. For instance, instead of growing an object like a list in a for loop, it is better to allocate the space required for the object before and modify each object's element in-place [96]. Another example is parallelisation which is supported by the built-in R package parallel [43, 96].

4.2.3 R Packages

R has a set of conventions to create packages that are easily installable in other computers. One of the best reference books about R packages is [98]. This book is divided in chapters according to the organisation of an R package describing each folder and file at the top-level of a package:

- `R/` folder for code (i.e. R files)
- `DESCRIPTION` file for package metadata including package description, dependencies, licence, file loading order and author contact information
- `NAMESPACE` file for a list of exported functions (i.e. functions meant to be used by end-users) and functions used from other packages
- `man/` folder for object documentation
- `tests/` folder for unit tests
- `vignettes/` folder for package tutorials and guides
- `data/` folder for data to be used by the end-users
- `src/` folder for compiled code (like C++ code)
- `inst/` folder for extra files like external data not to be used by the user, citation information and non-R source files

The best friend of an R package developer is the package devtools, given that it has dedicated functions to test the package, properly document the package and its functions, run the examples given in the function comments, create package tutorials and guides (known as vignettes) and build the package itself [46].

Code

R packages require files to be stored directly in the R folder to be tested, documented and built in the package. Any files outside this folder or even inside its subdirectories are not tested, documented and built in the package. This is a major problem regarding code organisation. To solve this, files have a shared prefix in its name to define their ”folder”. Still, this leaves the R folder containing many files.

R packages are distributed through a binary package where the functions from the R files are efficiently stored but the original source files are not available. R loads all those functions when loading a package.

Package Dependencies

There are two types of package dependencies: the packages that *must* be present for the package of interest to work (imported packages) and the packages that are not critical (suggested packages). Imported packages are automatically installed alongside the package of interest, while the suggested packages need to be installed by the user. It is also possible to indicate the minimum version required of a package. All package dependencies must be stated in the `DESCRIPTION` file.

Object Documentation

Object documentation is important to declare how functions work. Specifically in R, functions are documented by writing an individual file based on LaTeX located in the `man` folder [98]. Although creating a separate file in LaTeX for each function may seem too much work, the R package roxygen2 can handle this. With this package, the developer just needs to create roxygen comments (comments with special prefix and tags) before a given function and run a specific roxygen2 function to create or update the documentation files [50].

A roxygen comment for a function may include a title, a description, type and description of parameters, examples[3] and description of the function output. The available tags allow to identify these different sections and many more which can include formatted text like bold, italics and links. Documentation may be applied to datasets and to packages as well and, in the case of packages, it provides a way to describe the most important package components [98].

Documentation also allows to state which functions are accessible when loading the package (exported functions) and which are only for internal use (non-exported or internal functions)[4] [98].

By default, R loads all functions in files by alphabetical order of the files when loading a package. Although it is possible to change the loading order manually, roxygen2 makes it easier [50].

Vignettes

Vignettes are additional documentation more akin to a guide or a tutorial. Instead of describing a function or a package like object documentation, vignettes elucidate the problems that can be solved with a given package, discussing the most useful functions that are available and how to use them. Vignettes

[3] These examples are run by default to ensure the code is valid.

[4] Note that external functions are the ones that should be used by people who are using the package. It is still possible to use non-exported functions.

are written in Markdown, a plain text format that can be converted to HTML. The R package rmarkdown combines Markdown text, runnable R code and respective results [98]. Most of the work I developed was written down in vignettes, making it easier to explain certain steps of the package development.

4.2.4 Shiny

Shiny is an R package that works as a framework for web applications [19]. When starting a Shiny app, a localhost connection opens and is accessible through a web browser while the calculations are processed in an R session.

Shiny follows a reactive programming model where the output (for example, a plot) is bound to two specific inputs (for instance, the data to be plotted and the range of the X axis), so that every time the input is modified, the output is updated to reflect those changes in a so-called reactive dependency. All the input and output objects are recognised by a given unique identifier. Shiny makes writing the user interface easier by providing functions that are essentially wrappers to HTML code. Some functions are higher-level (e.g. it is easy to implement common input elements like radio buttons and checkboxes), but Shiny supports any kind of HTML tag.

Shiny also allows to directly include HTML, CSS and JavaScript elements in the application. HTML is a language used to structure the content of a page while CSS allows to style the HTML elements by their class, identifier and other attributes. JavaScript is the language used in web design to perform calculations and add interactivity to the web pages. It can also access the HTML elements by their attributes.

Shiny makes use of Bootstrap 3[5]. Bootstrap is a free and open-source HTML, CSS and JavaScript framework that includes many design templates for styling input elements, progress bars, navigation bars and other common elements in web pages.

The suggested folder organisation for a Shiny app includes the file(s) that starts the app, the user interface and server logic in the top-level directory, as well as a `www` folder containing the styling files (CSS) and external JavaScript libraries. However, this recommendation does not work for packages because custom top-level directories are not included after the package is built for distribution. Therefore, all R files are placed in the `R` folder while other files and folders are moved to the `inst` folder[6].

4.3 Requirement analysis

As previously mentioned in section 1.1: Requirements, the discussions with the stakeholders allowed to understand the requirements of the program, which were divided in the usual categories — functional and non-functional. The following list presents the functional requirements:

- Retrieve data (like clinical information and molecular data) from online sources such as TCGA
  - Ability for an administrator to allow data retrieval from other databases
- Unarchive downloaded data if needed (as in the case of TCGA data)

[5] `http://shiny.rstudio.com/articles/css.html` (last accessed on 1 September 2016)
[6] `http://shiny.rstudio.com/articles/html-ui.html` (last accessed on 1 September 2016)

- Identify and load data according to their file formats
  - Ability for an administrator to add support for new file formats
- Manipulate data (including alternative splicing quantification from data) input
  - Ability for an administrator to create new data processing tools
- Perform statistical analyses (e.g. survival, principal component and differential splicing analyses) and plot associated graphical elements (with a focus on user interactivity), saving results
  - Ability for the end-user to save results
  - Ability for the end-user to also create clinical groups to use in the analyses
  - Ability for an administrator to add new data analyses and visualisations

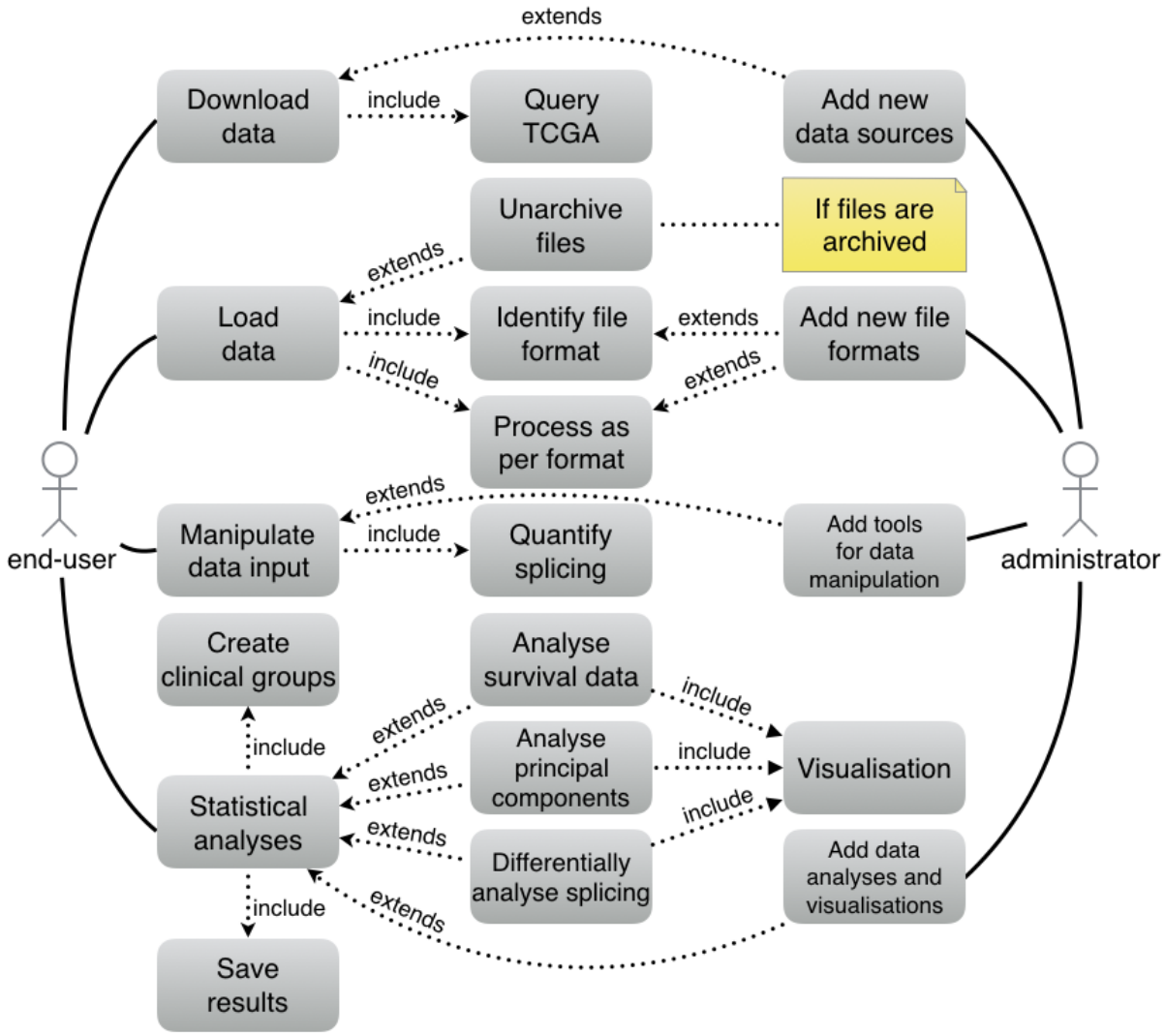


Figure 4.1: System use case diagram. Syntax based on Unified Modeling Language (UML).

These functional requirements were organised in the use cases presented in figure 4.1. The following list presents the quality attribute requirements:

- **Modifiability** — easy to modify and introduce new system components, such as adding support for new file formats and for new analyses and visualisations;
- **Usability** — easy-to-use and consistent interface, with informative error and warning messages;
- **Performance** — focus on time taken by each operation, given the amount of data to process and analyse;
- **Responsiveness** — inform the user if an operation is taking place (for instance, show task progress and disable the button that starts an action during a task).

4.4 Architecture

An important step of software design is the creation of an appropriate architecture, more easily achievable with a careful reflection on the program's requirements (section 4.3: Requirement analysis). In turn, the architecture promotes discussions about the best approach to follow with the stakeholders that more closely aligns with the requirements [99].

One of the first steps taken was to research previous architectural design implemented in R to be aware of any major difficulties I could find. I actually did not find any research article related to R architecture, although other functional languages such as Haskell have literature on the topic. I initially proposed an architecture based on an hierarchy of file sourcing where the modules would load their sub-modules but this could not work as an R package because packages are binary and there is no access to their source files, only to its functions. This approach was inspired by how other languages work and reflected how little I knew about R package development and reinforced the importance of properly knowing a language and the respective developmental process before planning its architecture.

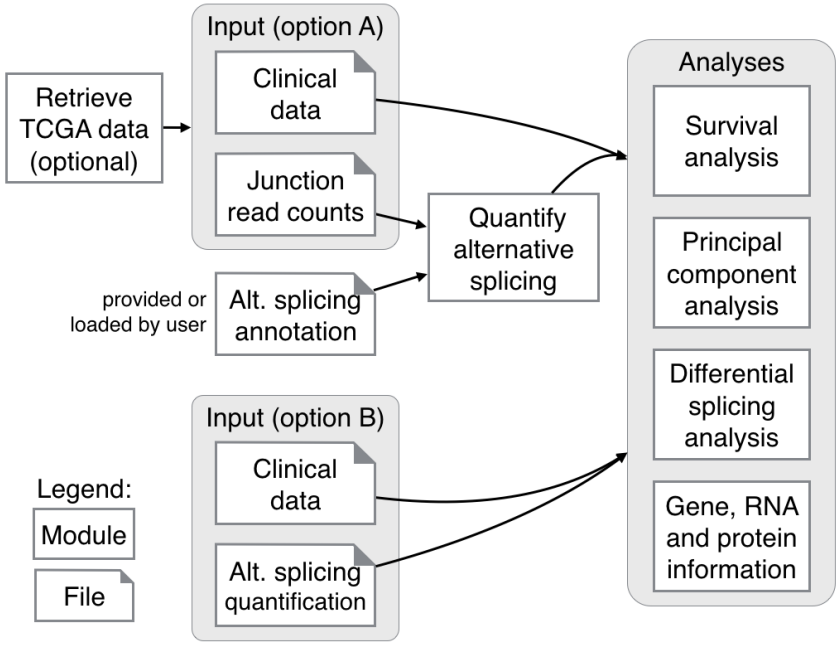


Figure 4.2: Logical view. The user has three options of data input: to retrieve TCGA data within the program, with the annotation of alternative splicing events either loaded by the user or provided by the program (a); to directly retrieve data and load it in the program (b); or to directly load the quantification of alternative splicing events, bypassing the loading of junction read counts (c). Each module comprises one or more files designed for a dedicated activity.

Based on the requirements, the logical view in figure 4.2 depicts how the application was designed to work from the end-user's perspective.

Given that modifiability is desired to easily extend and introduce new analyses and visualisations, the application was designed to be modular (i.e. to comprise independent components, which also makes unit testing easier) and extensible in order to allow the introduction of new features with little or no change of the program's core functionality. With knowledge on R and Shiny, a developer can follow simple template files to create new modules for the application.

Modifiability promotes the evolution of the application by making it easy for any fellow programmer (and maybe even biologists) to collaborate on new functionalities, thereby increasing its interest to the scientific community. Also of interest, the use of functional programming may contribute to the application's desired modularity and effortless testability as it characteristically reduces the problem into easier-to-tackle sub-problems [100]. Both modularity and extensibility contribute to a more maintainable program by making it easier to apply modifications or bug fixes.

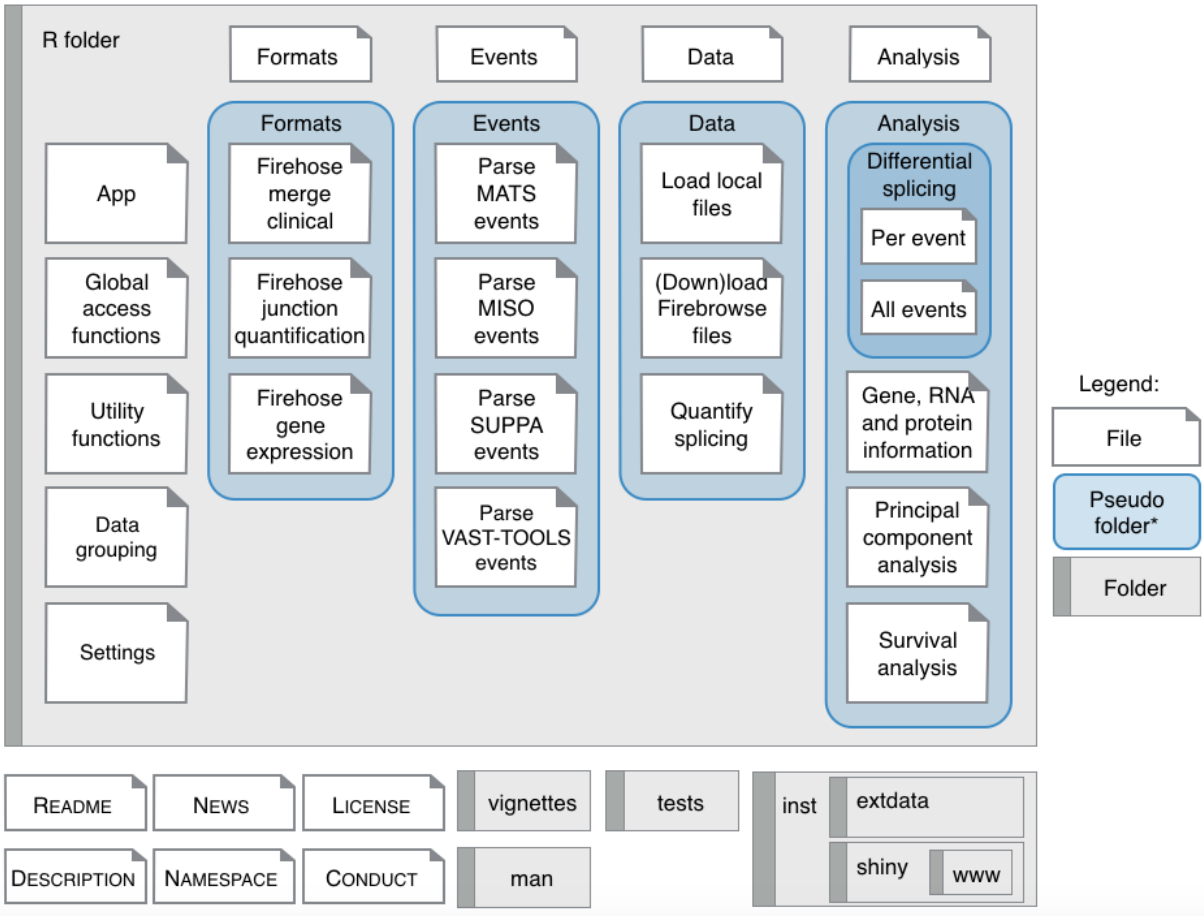


Figure 4.3: Development view. The top-level directory organisation of the program is conditioned by the structure of packages in R (see subsection 4.2.3: R Packages). Pseudo folder refers to the fact that source files cannot be inside subdirectories. As an alternative, filename prefixes are used to indicate the "folder" of a file. The pseudo-folder *Data* encompasses data retrieval and manipulation.

The most desirably modifiable parts of the system were mentioned in section 4.3: Requirement analysis: any interested developer should be able to add new modules to retrieve data from new online databases (**data retrieval**), support new file formats (**formats**), create new data processing and manipulation tools (**data manipulation**) and perform new data analyses and visualisations (**analyses**). In order

to do so, the system is decomposed in three parts: data input and manipulation (given their similar implementation), data loading and data analyses and visualisations. Also, to provide a complete alternative splicing annotation to the user, a collection of functions were created to parse the output from the different commonly used programs for alternative splicing analysis (**events**; read subsection 3.2.1: Alternative Splicing Annotation). This parsing is not part of the program's runtime. In the future, this may be used to cross-reference alternative splicing events shown in the application with those from other programs. Figure 4.3 depicts the modular organisation of the program's code.

Another important consideration is usability as a good user interface is crucial to user satisfaction. This program is intended to be used by anyone interested in studying alternative splicing, particularly researchers with no computational background. Given that one of the main goals of this work is to build a tool to be used by the scientific community, the application needs to provide what the users of this field expect in an intuitive way to be successful. This includes finding ways to help the user select data and display them in an interactive and intelligible manner.

All of these attributes were taken into consideration while designing and implementing the application. Following many discussions with the stakeholders about the analyses and interface and through an iterative development, we agreed on the architecture illustrated in figures 4.2 and 4.3 and further described in the remaining sections of this document.

Chapter 5

Implementation

The proposed application tries to satisfy any interested user that desires to study alternative splicing in cancer. The application contains a graphical user interface through which users can load, process, filter and analyse tumour data available from online databases in an easy and intuitive manner.

5.1 Modularity

Recently, Shiny was updated to standardise the usage of independent modules. These modules are comprised by a server logic and an user interface that can be called by other modules. Each module also employs its own namespace to avoid conflicts when sharing identifiers for input and output objects in distinct modules.

Before the introduction of Shiny modules, I was working on a similar implementation based on R environments (each environment is a collection of independent variables). My implementation ensured a hierarchy of environments where each submodule loaded its functions to an environment encapsulated in the environment of the calling module. This actually had a pretty significant improvement over Shiny modules as functions from different environments could have the exact same name, given that R does not allow function overloading per environment. However, this is not applicable to packages as, when an R package is loaded, all its functions are loaded overriding homonymous functions. Since both implementations were similar in this context, the implementation of the Shiny modules was preferred for being the standard.

The architecture of the program was designed to have a primary function called by the user that calls all the user interface and server logic of each complying module assigned to the primary function. In their turn, these modules call the functions for which they are responsible, and so on.

To assign which functions call which, each function has an extra attribute stating its caller function. This is possible given that any R object (including functions) is a container of publicly-accessible attributes, allowing to develop the mentioned hierarchy by attributing a string recognisable by a specific user interface or server logic functions. The caller's user interface function will call all the user interface functions with the given string and the same for the server logic. Also, a *prioritise* functionality was incorporated to change the loading order of functions (by default, functions are loaded alphabetically).

New modules can be easily added by following the existing templates without the need to modify any other file. For instance, check the code in section A.2: Template for an Analysis File.

When the user loads the project's package, they can either use the exported functions in the command line or launch the visual interface to use the program's functionality. The visual interface launches in the user's default browser by calling the function `psichomics()`[1]. This function also triggers a series of hierarchical calls to prepare the interface and server logic of the application (figure 5.1).

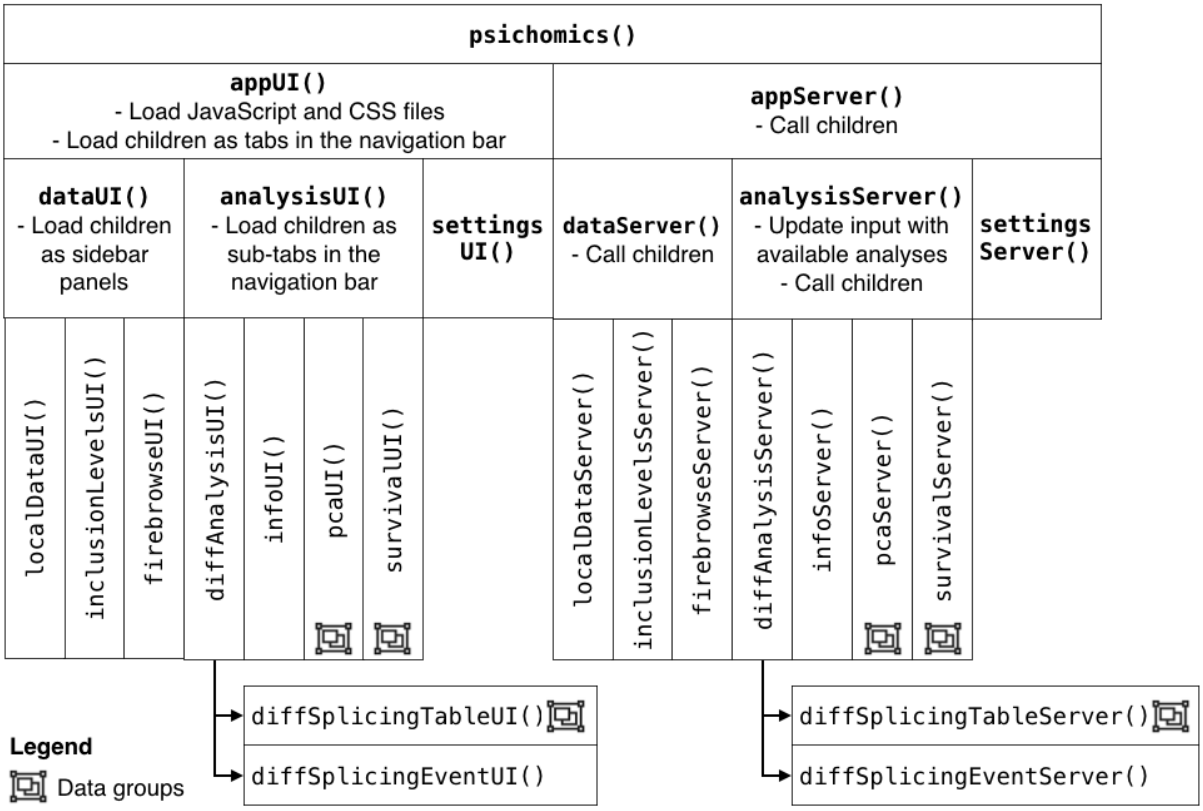


Figure 5.1: Call hierarchy for the interface and server logic functions. The call hierarchy is triggered by calling `psichomics()`. The icons represent functions called to create data groups (either the interface or the server logic, depending on the caller function).

As the functions for the user interface and server logic can be included in a single file, I decided against decoupling them in order to simply have one module's responsibility for each file. Since files in the R folder cannot rely on subdirectories to help in the organisation, I found it more appropriate to use one file to represent one module. Still, it is actually possible to use separate files.

5.1.1 Communication

All the variables are accessible through a special global reactive variable, specific to Shiny, that holds all the information during the runtime of the program. This variable is a list containing the loaded datasets, groups and global configurations. In order to abstract how data in the global variable are retrieved and set, dedicated accessor/getter and mutator/setter functions were created, being located in the file specific to global access functions (figure 4.3).

[1] As previously mentioned, PSIchomics is the name of the project.

5.2 Data Input

As previously mentioned in section 3.3: Alternative Splicing Quantification, the data required for the quantification of alternative splicing events are the junction read counts from the samples of interest and the alternative splicing event annotation. The latter is provided in the package, although only for Human (hg19 assembly). Besides, the user may also load the alternative splicing quantification previously calculated with this program.

The following subsections discuss how the data required for alternative splicing quantification are retrieved and loaded.

5.2.1 Junction Read Counts and Clinical Data

TCGA is a project that aims to collect data from patients with diverse tumour types. TCGA contains tumour transcriptomic data, including junction read counts and patient clinical information. Most of the data from TCGA is freely accessible through a RESTful service which allows to query and download the available datasets[2]. However, data from TCGA is organised as one file per clinical sample, requiring merging the results of all these files per data type (i.e. type of molecular profile) before data processing.

Fortunately, Firehose hosts data from all the TCGA samples of each cancer type merged into a single file per type of molecular profile. There is an R package to programatically retrieve the data, called Firebrowser [20]. However, this package is not available in neither CRAN nor Bioconductor. Considering the project is going to be submitted to Bioconductor, it cannot depend on packages unavailable in these repositories (see subsection 4.1.1: Bioconductor's Package Guidelines).

Firehose data is also accessible through Firebrowse's RESTful service which allows to retrieve data per data type, tumour type and data's date stamp [65]. This service was used to retrieve TCGA data of interest like clinical information and read counts for alternative splicing junctions. Querying Firebrowse's RESTful service returns a JavaScript Object Notation (JSON) file containing links to download gzip-compressed tar archives (one archive per data type per tumour type). The service also includes MD5 files that are used by the program to check the integrity of the downloaded archives.

The program's interface allows to select tumour type (also known as cohort), data's date stamp, type of data to retrieve (e.g. clinical data and junction quantification) and the folder where data are stored. The type of data to retrieve is an actual simplification of two Firebrowse fields. Instead of having the user select the correct combination of fields to get the data, it allows the user to select data of interest directly. For instance, the user can just select "Junction quantification (RNA-Seq)" which appropriately queries Firebrowse for junction quantification from RNA-seq.

As Shiny is only able to sequentially download files and the interface becomes unresponsive when transferring large files, file downloading is handled by the web browser. However, this implies that the user must understand that the files are being downloaded and that the program needs to be notified when the downloads have finished to continue processing them. To inform this to the user, a modal dialog instructs on how to proceed when the downloads finish (see subsection 5.4.1: Modals).

If all the requested data are available in the user's computer, the downloaded archives are extracted to folders named with the respective tumour type and date stamp for organisation purposes. Finally, the

[2] `https://wiki.nci.nih.gov/display/TCGA/Web+Services` (last accessed on 10 September 2016)

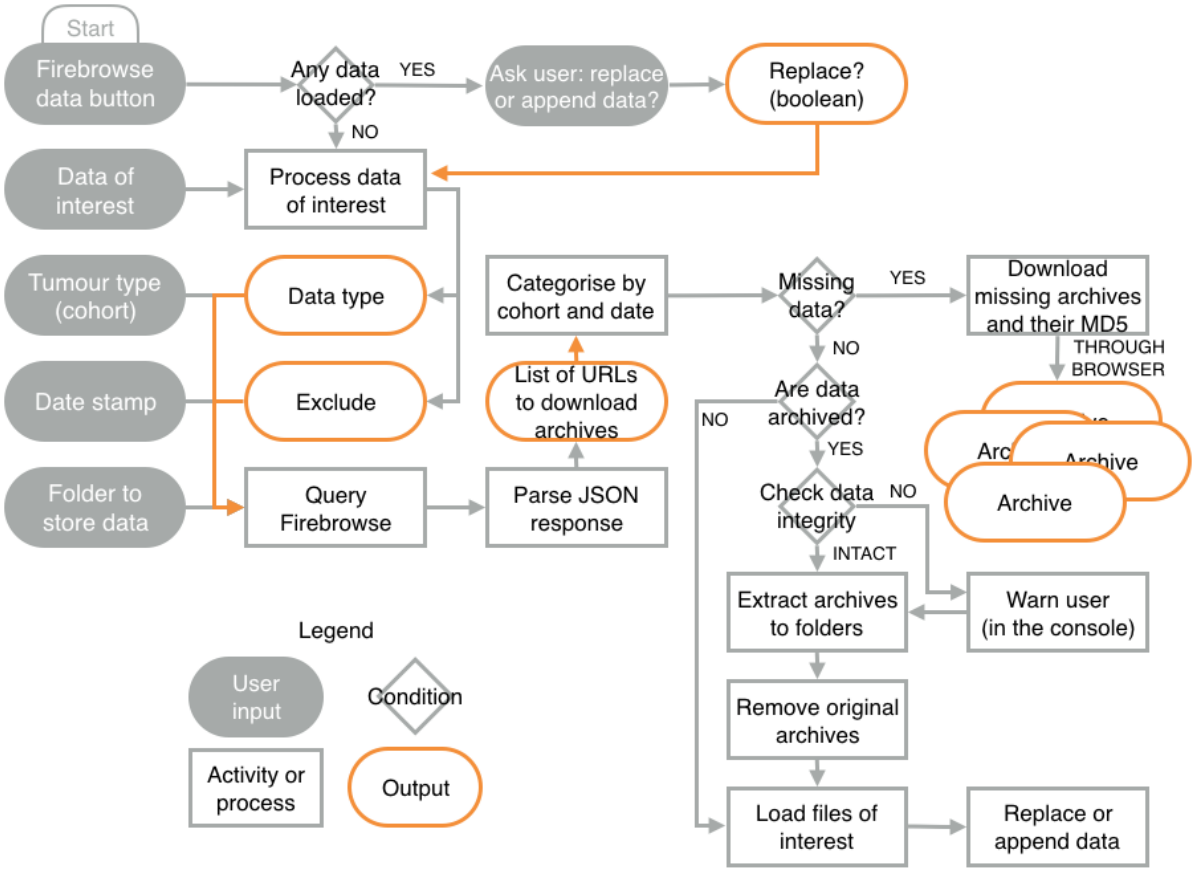


Figure 5.2: TCGA data retrieval. If any of the files required for processing is unavailable in the given folder, they are downloaded. Otherwise, files are loaded.

files inside those folders are loaded to the application.

Clinical information is contained inside a Tab-separated Values (TSV) file with multiple rows characterising each clinical patient (column) of a given tumour type, while junction quantification is available as a TSV file showing the junctions (rows) and the sample identifier (columns). As a patient may have more than one associated sample, multiple columns in the clinical information indicate the sample identifier. A simple search in the columns where sample identifiers are present allow to associate patients and samples.

5.2.2 Alternative Splicing Event Annotation

The human alternative splicing event annotation (hg19/GRCh37 assembly) provided in this program is retrieved from diverse programs: MISO [14], VAST-TOOLS [9,34], rMATS [15] and SUPPA [16]. The annotation files from MISO and VAST-TOOLS are available online while SUPPA and rMATS require to run their software using mRNA annotation — a file containing the genomic coordinates of all transcripts and respective exons, downloadable from the UCSC Table Browser [68].

Although it is possible to create the alternative splicing event annotation file with functions within the program, the visual interface does not accommodate this as it was planned that the annotation would be provided to the user (even though the user can still load his/her own annotation file).

Unfortunately, the inclusion of the combined annotation file makes the package larger than the 4

MB limit of Bioconductor (subsection 4.1.1: Bioconductor's Package Guidelines), so this file will be provided instead as an annotation package in Bioconductor[3].

5.2.3 Dataset Loading

Dataset loading progresses in two steps: (1) check the format of a file and (2) load the file according to its format's instructions (figure 5.3). Each format of interest has its own R file to describe its properties, including the header of the file, the name of the file and how to load it, among many other attributes, in order to accommodate a wide range of file formats. For consistency with the remaining project, the language of the source code is R, instead of XML, JSON or a custom domain-specific language. The source code that defines a file format contains a function that returns a named list (also known as *associative array*, *dictionary* and *key-value pair* in other programming languages). This list determines the file format's properties (see the source of the clinical data format in section A.1: Format of the Clinical Information).

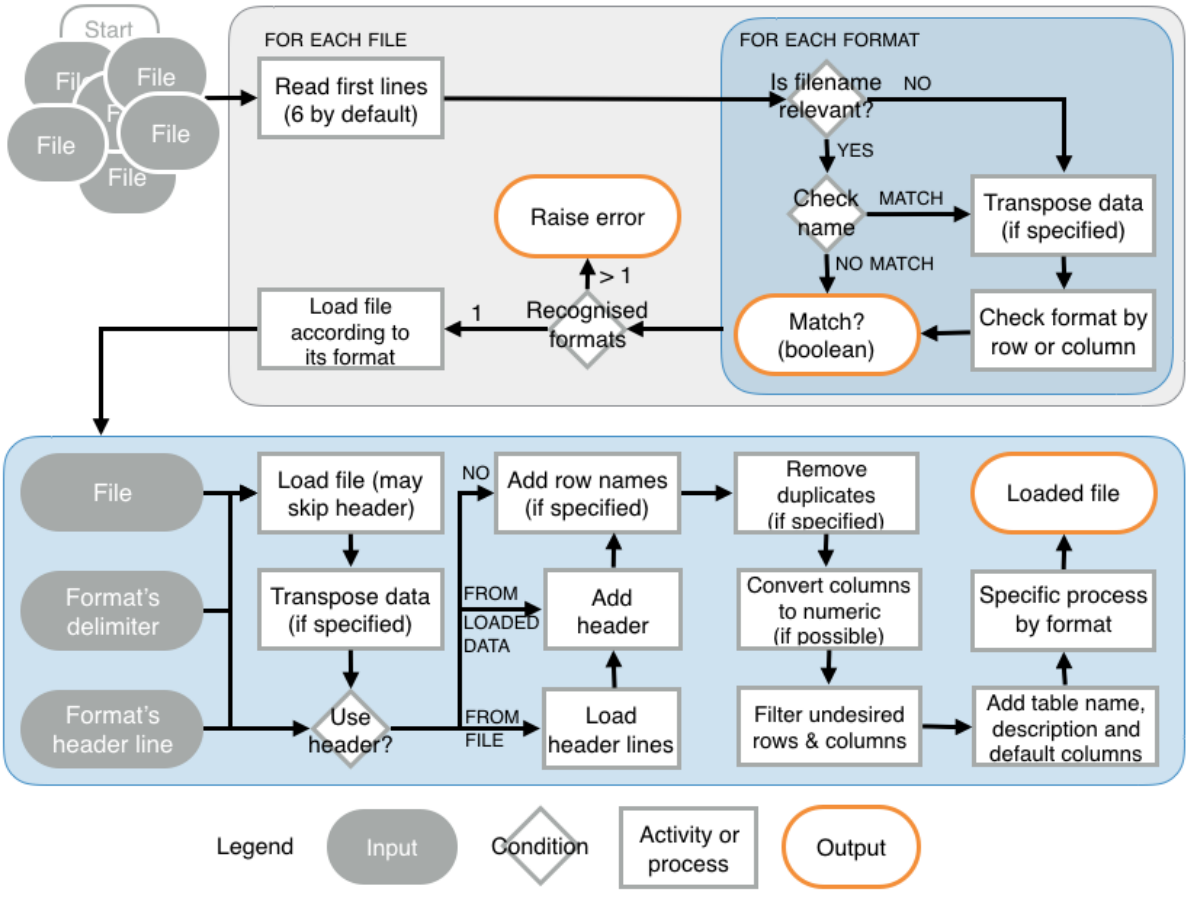


Figure 5.3: File format checking (top) and file loading (bottom). Each file is checked against the available file formats. If only one format returns positive for a file, that file will be loaded according to the format's instructions.

When checking the format of a file, only the first lines[4] of the file are loaded. A file is matched to a

[3] Annotation packages do not have size limit.

[4] By default, it loads the first 6 lines of the file. This can be changed in the format's attributes.

format if the format's string and a specific row or column of the file are consistent. Optionally, a partial match for the filename may be additionally requested. Currently, only three file formats from TCGA are loadable by our program: junction quantification, gene expression[5] and clinical data files.

If a file's format is identified, the file is loaded according to the matching format's instructions (bottom of figure 5.3). R loads files by automatically recognising each column type (string or numeric) but this does not work when the file has a header made of strings. So, instead of loading the whole file, removing the rows containing strings, adding the desired row as a header and converting all the R object's columns to numbers, it is an order of magnitude faster to load the file without those first lines with strings and then load what was skipped and use it as the header. Also, duplicated rows can be removed. This is faster if row names are available, as they are required to be unique and duplicated values can immediately be identified. However, if a column of unique values is not available, rows need to be compared one by one in a much slower process.

5.3 Analyses

5.3.1 Interactive Plots

An important tool to create interactive elements in Shiny is the package htmlwidgets that makes it easy to wrap JavaScript libraries in R [101]. Currently, about 70 htmlwidgets-based HTML widgets are hosted in CRAN[6].

Many R packages were explored to look for a package able to create diverse types of different plots (for example, able to create general plots like bar and scatter plots and more specialised plots like heatmaps and survival curves). Having a package able to perform a wide number of plots allows for interaction consistency (for instance, different plots share the same way of zooming) and is easier to develop with (each package has their own functions and it can be time-consuming to learn how to plot different charts in the different packages).

ggplot2 is a very popular and extensible plotting system in R to create diverse types of static plots [102]. ggplot2 comes with some rather basic functionality for user interactivity in Shiny apps. It includes functions to retrieve the location and data associated with the element hovered, clicked and double-clicked by the user and also a brushing feature that allows the user to draw a rectangle selection on the plot. It can even zoom and return points near a mouse event or a brushing area [102].

Many packages extend ggplot2's functionality and such is the case with ggiraph and ggvis, that include contextual tooltips and support JavaScript click events on data points but are limited to few types of plots. plotly is another package that also extends ggplot2 and natively generates scientific plots such as survival curves and heat maps. When the appreciation of these graphical tools was made, the use of plotly required authentication and an internet connection to publicly upload plots (with a limited number of private plots for the free account) which made me put the package aside. This is not true anymore since plotly can now be used independently of authentication and online access.

rCharts is an R package that aggregates many different JavaScript libraries which do not share much consistency between them (as already mentioned, it is more time-consuming to develop using different

[5] Gene expression is presently not supported in subsequent analyses.

[6] See `http://gallery.htmlwidgets.org` (last accessed on 4 September 2016)

libraries and the interactions are not the same between plots of different libraries). An even bigger problem of this package is the scarce documentation available, which limits its potential.

d3heatmap is an implementation of a heat map including a dendrogram based on the D3 JavaScript library. Although it only plots heat maps, this type of interactive plots are overlooked by other packages. Besides, it allows zooming and contextual tooltips on hover. Another interesting package is edgebundleR that only creates interactive circle plots.

Metricsgraphics is based on MetricsGraphics.js, a JavaScript library built on top of the D3 JavaScript library. It features pretty plots, customisable contextual menu on data hovering and seamless data updating. However, data zooming is an add-on and notably slow. Besides, MetricsGraphics.js only focuses on line charts, even though the experimental scatterplots, bar plots and column plots are nicely done[7].

There is an R package named Highcharter which is a wrapper to *Highcharts*, a JavaScript library free for non-comercial use [18]. *Highcharts* supports tooltip on hover, zooming, omitting series and exporting the plot to common image formats such as PNG, among other features. More impressive yet is the array of plot types available through *Highcharts*: line, area, column, bar, pie, scatter and bubble charts, heat and tree maps, boxplots, polygon series and it even allows free drawing[8]. All these charts are highly customisable and available in the R package. For this reason, Highcharter was chosen as the primary plotting library to render most interactive plots.

5.3.2 Principal Component Analysis

The current implementation of the PCA module allows to standardise and scale the data, impute or remove missing values according to a given threshold and perform PCA on the alternative splicing quantification data to explore groups between data samples.

To perform a PCA, the program uses the function `prcomp` from the built-in package stats, resulting in an object of the class `prcomp`. As already mentioned, R objects are essentially a list of attributes and it is easy to retrieve information from this object to render a scatter plot using two principal components (X and Y axis) with Highcharter. Unfortunately, adding attributes to the individual data points (such as the sample identifier in this case) was not available in Highcharter so a modification to an existing function in Highcharter was sent as a pull request and accepted. It is now available in the lastest release of Highcharter in CRAN.

It is also possible to colour samples according to their clinical attributes (e.g. tumour stage, gender and race) by creating groups of data. More information is available in subsection 5.4.2: Data Grouping.

To easily visualise the principal components, the proportion of variance explained by each principal component are represented both graphically and textually.

5.3.3 Survival Analysis

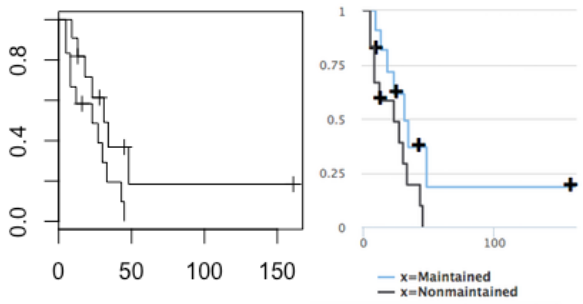
The survival analysis module allows to analyse patient survival using Kaplan-Meier curves and by fitting a Cox proportional hazards model. These two analyses have dedicated buttons, given that the input they receive may be slightly different. The statistical significance of differences between Kaplan-Meier curves is also calculated when plotting the curves using the `survdiff()` function from the survival

[7] `http://metricsgraphicsjs.org` (last accessed on 4 September 2016)

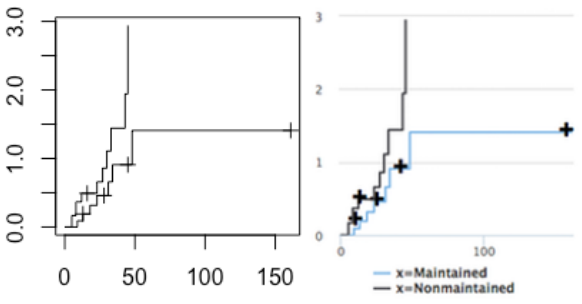
[8] `http://www.highcharts.com/demo` (last accessed on 4 September 2016)

package and information associated with each curve (number of patients and events) is available in the plot's tooltip.

The output of the survival analysis using follow-up time (or start and end times in case of interval-censored data) and event occurrence through the `survfit()` function (survival package) is an R object of the class *survfit* which can be used as input to the function `plot.survfit()` for a static plot. To create interactive plots, the function `plot.survfit()` was studied and simplified so the R object could be plotted with Highcharter instead (figure 5.4). The code for the Highcharter plot was submitted as a pull request and accepted in the development version of the Highcharter package.



(a) Static (left) and interactive (right) Kaplan-Meier curves



(b) Static (left) and interactive (right) cumulative hazard curves

Figure 5.4: Comparison of survival curves comparison as plotted using the function `plot.survfit()` from the package survival and Highcharter.

To calculate the fit of a Cox model to the clinical data, both follow-up time and event occurrence are passed to the `coxph` function from the survival package. The follow-up time is simply the time when an event occurred or otherwise the time when the patient has last been observed (censored).

To model the terms of the Kaplan-Meier curves or the Cox model fit, the user can visually create groups based on the clinical data (say by tumour stages or ethnicity) to analyse differences in survival (see subsection 5.4.2: Data Grouping). More advanced users may prefer to insert an expression to analyse survival data (as it is usual in R) to explore interactions between clinical groups for Cox models. When inputting a formula, the clinical attributes available are accessible through text suggestions (see section subsection 5.4.3: Text Suggestions). The expression is written in a text box and the resulting string is parsed to run in R. In case there are parsing errors, the user is notified.

Survival analyses can also be used to study the impact of alternative splicing in prognosis using the PSI cut-offs to group samples based on an alternative splicing event. Using the R built-in function `optim()`, multiple cut-offs are tested for each selected event and the one maximising the significance of difference in survival (i.e. minimising the p-value of the log-rank test) between the two groups is returned. However, it should be noted that although the `optim()` is relatively fast, it may return a local *maximum*.

5.3.4 Differential Splicing Analysis

Differential splicing analysis has been divided into two submodules. The first one is responsible for performing statistical tests on the PSI differences between groups of samples for all the alternative

splicing events and returns a table with the relevant results. This table can be saved by the user as a TSV file. Contrastingly, the other module displays the results of the statistical tests for each individual event.

Both submodules are accompanied by density plots so the user can observe the distributions of alternative splicing quantifications for each group and visually compare them (figure 5.5). To avoid difficulties in interpreting low-resolution density plots resulting from small sample sizes, the data points may be plotted near the axis as a visual guide for the user to broadly understand the variance and number of samples contributing to each curve. This plot is known as a rug plot.

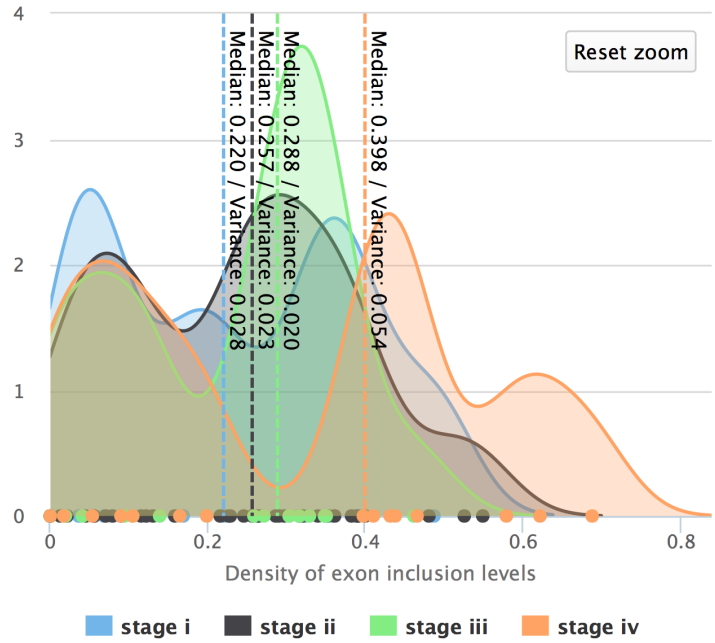


Figure 5.5: Example of a density plot containing a rug plot at the bottom. The distributions of alternative splicing quantifications for this splicing event over the different tumour stages show an increase of the median value (represented by the vertical dashed line) with maligancy.

Highcharter does not possess any function to render density plots but they are rather easy to create. The code to create them was sent as a pull request to the GitHub repository of Highcharter and it is now available in the latest release.

The table containing results of the statistical analyses for all the splicing events can be filtered by their values or the spicing events' attributes (e.g. chromosome, strand and associated gene). It also possesses a column with the density plot estimated with 10 points from each group for each splicing event for performance reasons (compare this with the 512 points used by the density plots for single events). The splicing event, the identifier and the density plot are both clickable and take the user to its differential splicing analysis, allowing the user to focus on the statistical information available for that event alone.

The density column in DataTables does not use the Highcharter package as it does not support rendering plots inside DataTables. In order to circumvent this limitation, I had to rethink my approach. As *Highcharts* transforms JSON code to the JavaScript-based plots, Highcharter is used to obtain the JSON code for the plots. Next, the JSON code is injected in HTML code as a string and added as a column of the table. The table has an associated callback to render the plots of the visible rows before drawing those rows on the screen.

5.3.5 Gene, Transcript and Protein Information

This section of the program allows to retrieve information from the gene, its transcripts (i.e. mRNAs) and respective proteins for the selected alternative splicing event. The information is concise, showing the gene name, a brief description of its function and its genomic position. Schemes of its transcripts and protein domains are also plotted. It also features links to appropriate external biological databases

such as Ensembl (genomic information [103]), UniProt (protein information [104]) and UCSC Genome Browser (genomic information visualiser [105]) based on the gene coordinates or the protein identifier.

Also, relevant articles from PubMed Central (a central repository of literature in life sciences [67]) are retrieved based on the gene symbol (a gene's unique identifier), the term *cancer* and the names of the datasets loaded by the user (e.g. *breast invasive carcinoma*).

All the information presented therein is retrieved from the Ensembl and UniProt RESTful services as depicted in figure 5.6. The user-selected alternative splicing event's gene symbol is used to query Ensembl's RESTful service for more information. Ensembl returns information regarding the gene, its transcripts and the transcripts' respective proteins [66]. This Ensembl protein identifier is matched against UniProt to retrieve protein features such as domains and structure [104]. UniProt proteins from both TrEmbl and Swiss-Prot are retrieved.

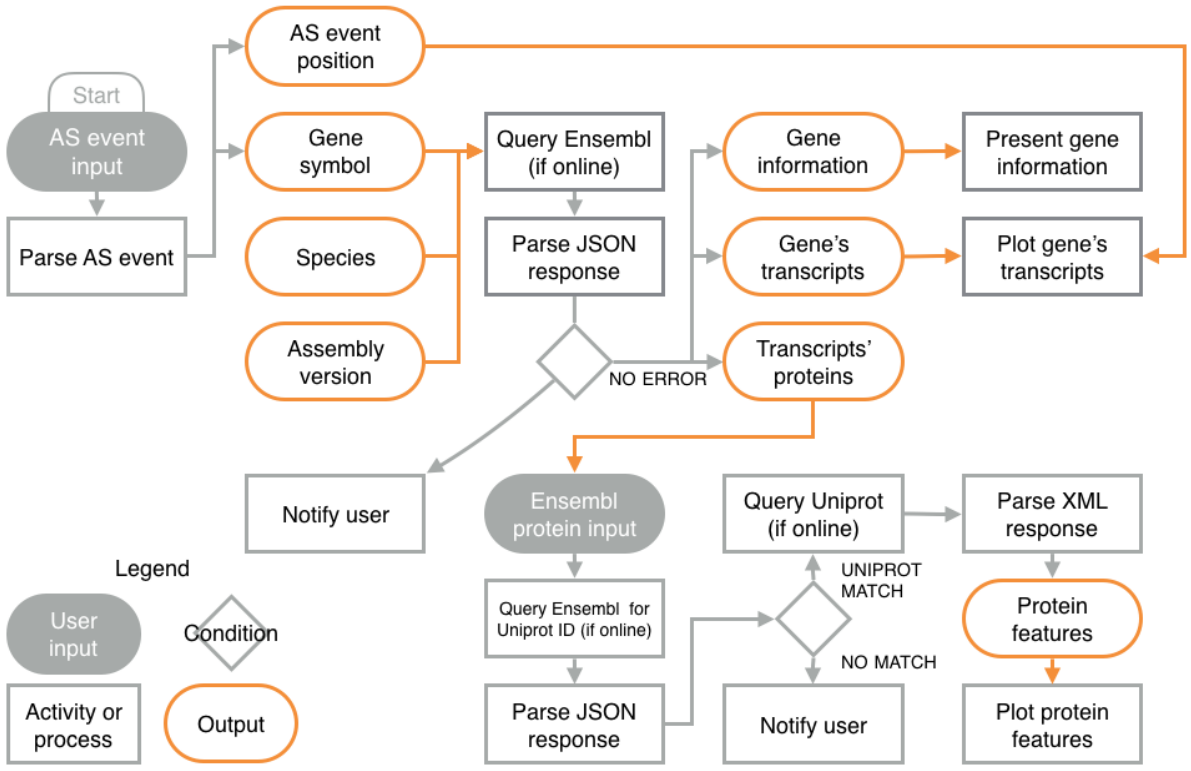


Figure 5.6: Process view of how the information on genes, transcripts and proteins is retrieved from Ensembl and UniProt RESTful services.

One of the first ideas for this section included embedding an HTML5 genome browser such as Dalliance, which uses Ensembl as one of its databases of reference [106]. This way, the browser view could be set to the gene coordinates but the user could still see surrounding genes, transcripts and proteins by dragging or scrolling the view. Unfortunately, the HTML5 or JavaScript genome browsers found either do not work in Shiny (such is the case of Dalliance) or do not provide a customisable and simple interface.

The transcripts are currently drawn using the R package Sushi, which results in static plots representing different transcripts (figure B.7b). Sushi requires as input the transcripts' identifiers, chromosomes, strands and the respective exons' start and end positions. Sushi is also able to draw a region over the

alternative splicing event of interest, allowing the user to check if the event is supported by transcripts from Ensembl. Unfortunately, trying to plot the same transcripts but altering the X axis to show only the alternative splicing event shows no transcript unless the transcript's start or end coordinate is within that region. This seems like an odd oversight and I find it an incentive to develop an interactive plot to replace the static one currently used.

To draw the protein features, the R package Highcharter was used. Only one protein can be selected at a time to have its features drawn. Each feature type from the protein returned by UniProt is represented as a separate series (figure B.7c). This way, the user can omit or show types of features from the plot as desired. For visual differentiation, each series has its own colour[9] and Y value. The differing Y value avoids overlaps between multiple data series.

The differing types of features also hold characteristic information. For instance, some features indicate an annotated change in a protein, its type and a brief description, while other features only present the description. To check this information, the user needs to hover over the feature of interest and the tooltip will present all the information available from the UniProt RESTful service regarding that particular feature.

Welcome future additions include a feature comparison between different proteins and highlighting the protein region affected by the selected alternative splicing event to better understand the protein features potentially affected by it.

5.4 Supporting Features

There are other supporting software features to ensure the application offers a consistent and intuitive interface. Some features such as explanatory progress bars are necessary so that thee user understands the steps and progress of long processes. Whenever the application is loading, processing a file or performing some time-consuming analyses, the application shows a progress bar indicating the level of progress of the current task.

Complementarily, a rotating science flask appears accompanied by the word *Working...* every time the application is busy, unlike the progress bar that needs to be explicitly called by time-consuming processes. When R is performing calculations, Shiny adds the class `shiny-busy` to the `html` tag; otherwise, it removes that class. The mentioned flask icon has a JavaScript condition to show or hide itself depending on that class attribute. When it is active, it can be spotted rotating in the top right of the page thanks to some CSS styling and the rotating animation provided by FontAwesome[10]. Since the rotating animation of the icon is based on CSS3, the animation is not supported by Internet Explorer 9 or older[11]. One drawback of this approach is that the class `shiny-busy` is sometimes removed before the view is completely updated, leaving the user with no indication of progress.

There are other relevant features that will be described in more detail: the modal dialogs to convey or prompt for information, the data grouping to create and edit groups and perform set operations like merge and intersect, and the text suggestions to show contextual recommendations on text input fields.

[9] Notice that Highcharts only provides 10 colours for differentiating series by default which means that assigned colours repeat when there are more than 10 series as it is shown in figure B.7c

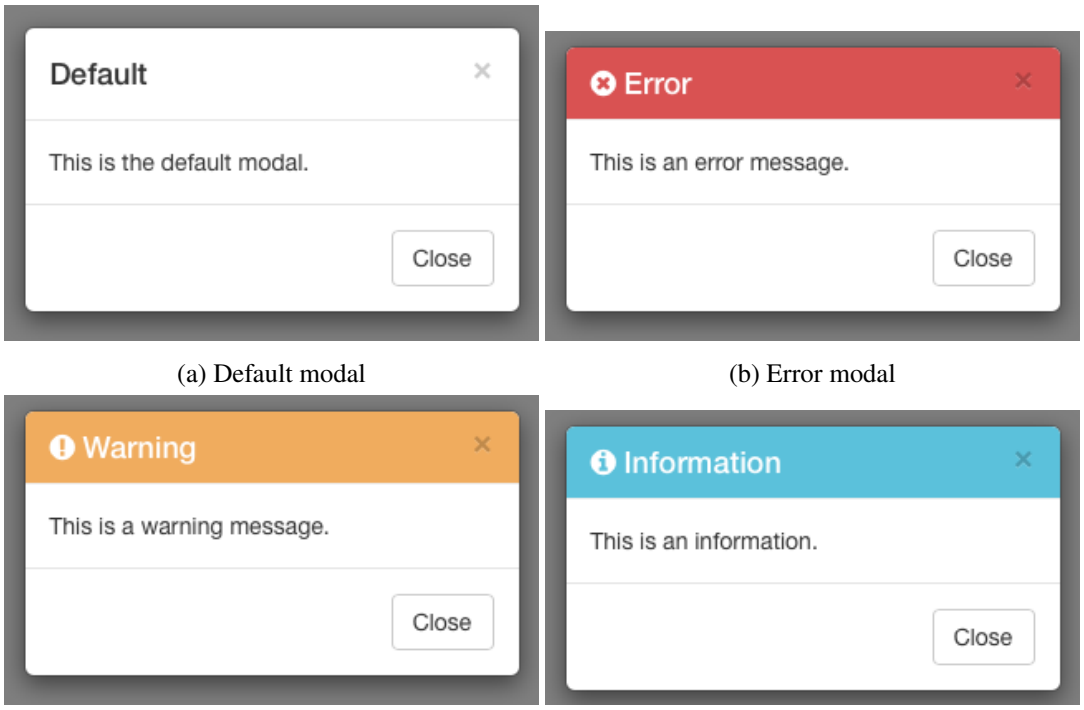
[10] `http://fontawesome.io` (last accessed on 15 September 2016)

[11] `http://caniuse.com/#feat=css-animation` (last accessed on 21 September 2016)

5.4.1 Modals

Modals are JavaScript dialogs that obscure the remaining content so that the user focuses on the displayed message. This is useful for user interactions such as when informing the user, raising warning and errors or as a data prompt. Modals have a header, body and buttons in the bottom and the body can contain any HTML content[12].

To make the modals visually distinct depending on the information presented, the modal dialogs were styled with CSS to different styles according to their HTML class attribute: warning, error and info (figure 5.7).



(a) Default modal

(b) Error modal

(c) Warning modal

(d) Information modal

Figure 5.7: Modal dialog styled according to the conveyed information.

5.4.2 Data Grouping

Data grouping is contextual. The interface to create and edit groups can only be triggered by a button near the input fields where groups are needed as it is the case for the principal component and survival analyses. When the user clicks the mentioned button, a modal dialog opens showing the interface to create and edit groups (figure 5.8). Created data groups are displayed inside the modal and within an interactive table.

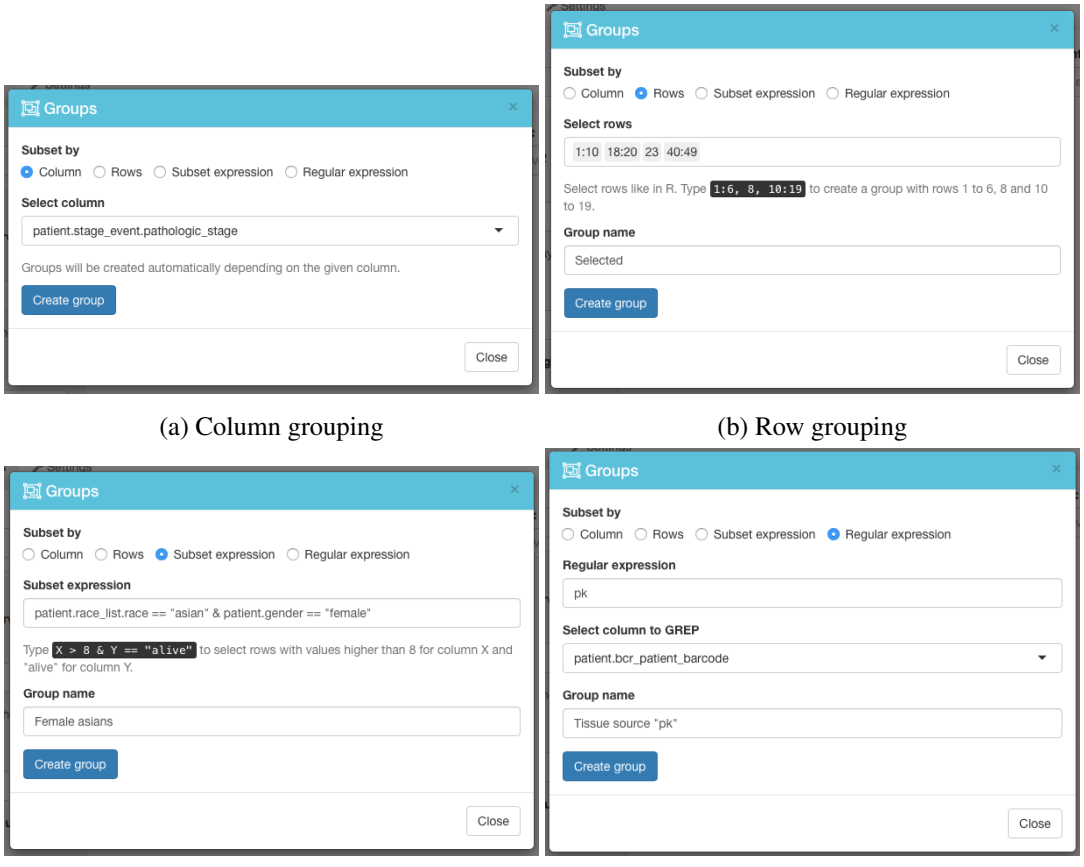
Each dataset holds their own groups list containing the group name, the row indexes and a reproducible description of the user input (so the user knows exactly what is represented by the group). A big limitation of this approach is that the datasets must not be modified row-wise unless the group indexes in the data groups are updated as well to reflect such changes. This was not implemented since datasets'

[12] `http://getbootstrap.com/javascript/#modals` (last accessed on 21 September, 2016)

rows are not supposed to be inserted nor deleted. However, this approach allows for fast set operations like merge/union and intersect since they only act on row indexes instead of rows of data.

Initially, data groups were positioned in the input tab but its location was confusing given that an explanation of how the groups would be used for downstream analyses was required. To solve this, data grouping is now contextual: any input field requiring data groups provides an additional button that allows to create and edit said groups. The *groups.R* file contains two functions responsible for showing the interface and running the logic on the dataset of interest which is passed to the server logic function as an argument. These dedicated functions make the elements of group selection consistent across the application and easily placed in any module requiring groups.

Data grouping allows to group rows of a dataset based on columns, row index, subset expression and regular expression (figure 5.8):



(a) Column grouping

(b) Row grouping

(c) Subset grouping

(d) Regular expression grouping

Figure 5.8: Data grouping

- If set by **columns**, the user selects a column from the dataset and groups will be automatically created based on the unique values of that column. For example, it is possible to group individuals by tumour stage which would create one group for the individuals at each tumour stage.
- In **row index**, the user can input the index of the rows of interest separated by commas (e.g. to indicate a group with the rows 1, 4 to 20 and 36, the user would input `1,4:20,36`).
- **Subset expression** allows to subset a dataset like in R using boolean operators and even running

functions if desired. The input field shows suggestions for the clinical data attributes to facilitate writing expressions (see 5.4.3 Text Suggestions).

- **Regular expression** allows to subset a dataset using regular expressions for a selected column of that dataset.

Each data group also needs to have a unique name. If the users attempt to set an already used name, the program will simply add a counter to it. For instance, when creating a group named *Tumour* when there is already a group with that name, *Tumour (1)* will be automatically used instead; then *Tumour (2)*, *Tumour (3)*, etc.

There are also safeguards in-place to show an alert when the users inputs a wrong expression or tries to get non-existing data (for example, trying to retrieve the 50th row when there are only 49 rows in the dataset). As previously mentioned, the group editing interface itself is a modal dialog. When an error or warning is raised, an alert is shown in the top. Alerts are used instead of modals because stackable modals are not supported and create a lot of issues.

To allow for set operations, the groups in the data table are selectable and three buttons are available below the table: merge/union, intersect and remove. These buttons are only enabled if any row is selected.

One of the first implementations of data groups included saving them as attributes of the loaded datasets. Although this seemed natural at the time, the reactive model of Shiny makes the table in the Data input section re-render. This is unneeded since the table as a whole does not change, just one of its attributes. This issue was easily solved by creating variables in the global variable with the same named of the datasets with the addition of a suffix like *Groups*. Anyway, since the access and manipulation of variables inside the global variable is abstracted through accessor and mutator functions, the implementation can be easily changed as desired.

One additional idea that was not implemented at the time of writing is row grouping[13] where rows of a dataset are displayed and sorted by group. This way, the user could visually browse the elements of created groups.

5.4.3 Text Suggestions

Text suggestions are provided in a dropdown menu in certain fields to help the user during text completion. Such completions are useful when the user is writing a dataset column name, for instance. In this case, the text suggestions can present to the user some suggestions to complete the column name from all column names available.

To make this more useful, the text completion's dropdown menu from the JavaScript library *jquery-textcomplete* was used with the fuzzy string matching from the JavaScript library *fuzzy.js*. Fuzzy string matching (more formally known as approximate string matching) scores the matches between two strings according to their edit distance. When the user starts typing in a field that supports text suggestions, each keystroke will show the highest-scored matches. The user can refine the suggestions by continuing typing. Selectize input elements in this application also support search by default thanks to *selectize.js* that

[13] `https://datatables.net/examples/advanced_init/row_grouping.html` (last accessed on 21 September 2016)

comes with Shiny, although its engine only allows searching using exact matching by default. However, its API allows to use another function to give the scores. In the future, the search function may be replaced by the one from *fuzzy.js* for consistency and for the improved results provided by this library.

5.4.4 Global Settings

There are settings to tweak the number of cores, numeric precision and significant digits used throughout the application using slider inputs. Since the reactive model of Shiny would trigger analyses to be redone with these values, it is best if the caller function does not take a reactive dependency whenever these values change to avoid performing potentially expensive calculations without the user's consent.

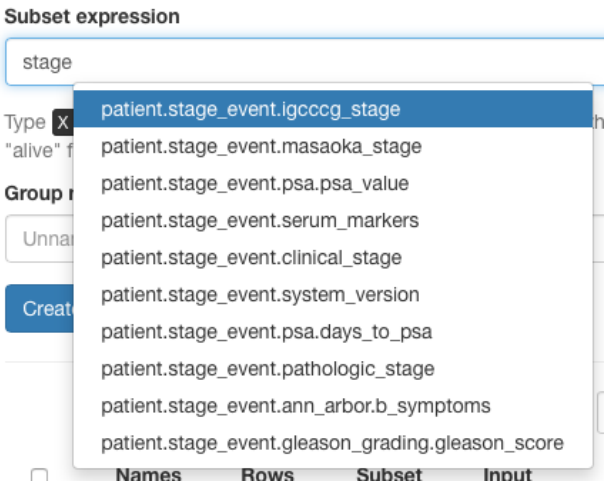


Figure 5.9: Text suggestions for clinical data attributes in the data groups modal.

The slider for the number of cores to use defaults to 1 and allows to choose up to the number of available cores. Currently, there is no parallelisation performed in the app, so this was developed for future use. However, there were attempts at parallelisation during the project. For instance, the statistical analyses for differential splicing were originally planned to be parallelised but running them in a Shiny session greatly impacts the computer's performance, even though the same commands work successfully when running directly in the R console. This is intended to be further studied in the future.

5.5 Case Study

To better illustrate the program's functionality, the following case study is provided as an example on how to analyse alternative splicing events from samples of patients with adrenocortical carcinoma.

Assuming the program is already installed, the user is required to open an R console, load the program using `library(psichomics)` and start the visual interface with `psichomics()`. These instructions are also available in the user tutorial. After the visual interface loads in the default browser, a welcome message is shown with instructions on how to load data, quantify splicing and filter statistically significant alternative splicing events (figure B.1).

To load data from Firehose, the user clicks in the *Load TCGA/Firehose Data* panel, selects the tumour type, sample date and data type (e.g. clinical data and junction quantification) and clicks on the *Load data* button (see figure 5.10). If the required data is not present in the given folder, a modal will appear informing the user that the archives are now being downloaded and that the user should click on the *Load data* button once the downloads have finished. This allows the program to automatically extract the content of the downloaded archives and organise the files within before loading the files.

The loaded datasets are presented to the user (figure B.2). Each dataset has its own tab named after itself which shows a description of the dataset, a download button[14] and an input where the user can select the visible columns. The datasets themselves are shown in interactive jQuery-based tables provided by the package DT, which allows the user to filter, sort and search the dataset [53].

Datasets like clinical data contain a large amount of information to render in a table. While rows are divided in pages to mitigate performance issues, a large amount of columns are slowly rendered on-screen. To avoid this issue, only pre-selected columns defined for a specific file format are rendered (read subsection 5.2.3: Dataset Loading).

Alternative splicing may be quantified by clicking on the *Quantify alternative splicing events* panel and selecting the junction quantification and alternative splicing annotation (see section 3.3: Alternative Splicing Quantification). The alternative splicing quantification is shown and is downloadable as any other loaded dataset.

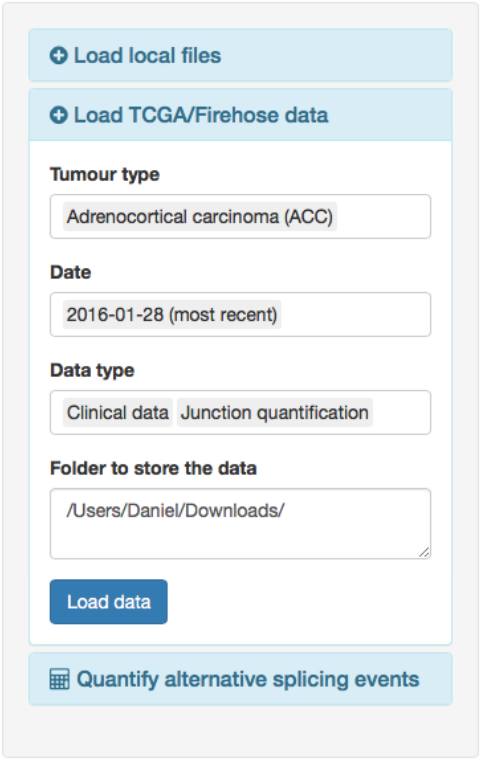


Figure 5.10: Options available to load data from Firehose.

Principal component analysis can be performed on the alternative splicing quantification to compare groups of samples. For instance, samples may be grouped by tumour stages. Tumour stages 1 and 2 seem to differ from stages 3 and 4 in samples from patients with adrenocortical carcinoma (figure 5.11).

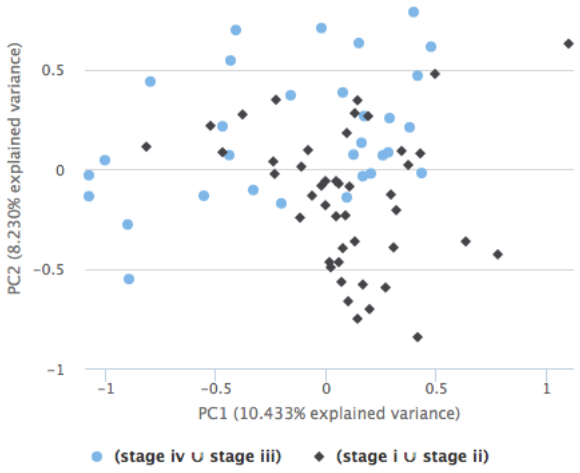


Figure 5.11: Plot of a principal component analysis of the alternative splicing quantification for samples of patients with adrenocortical carcinoma.

As there seems to be a difference between the mentioned groups, they were used for differential splicing analysis, using the available median- and variance-based statistical tests (read subsection 3.4.3: Differential Splicing Analysis). The output of differential splicing analysis is a sortable and filtrable table containing the statistical results for each alternative splicing event (figure B.4). To select statistical

[14] Although datasets can be loaded from the user's files, there are some that can be created from the input available. These datasets benefit more from the download button, which is kept in all datasets for consistency.

significant events, the p-values of the Wilcoxon rank sum, Kruskal-Wallis rank sum and Levene's tests were filtered to below 0.05 and the minimum number of samples per group was set to 20 samples.

To study the impact of alternative splicing events on prognosis, survival data can be incorporated. Kaplan-Meier curves can be plotted for groups of patients separated by the PSI that maximizes the significance of their difference in survival (i.e. minimizes the p-value of the difference tests in survival between individuals with PSI below and above that threshold). Given that calculating the optimal splicing quantification cut-off can be a slow process, survival analysis is only performed for the 10 events shown on-screen in the differential splicing analysis table by default.

One interesting alternative splicing event is a skipped exon with the identifier *SE 17 + 48624646 48625026 48625128 48625644 SPATA20*. Its survival curve shows the minimal log-rank optimal p-value of 0.0197 (which can be considered significant) and separates 34 (with PSI values below 0.89) from 19 patients (with PSI values higher or equal to 0.89). By clicking the identifier of the event above the survival curves, the user is taken to the survival analysis page where the threshold used to calculate survival difference is modifiable, among other options (figure B.6).

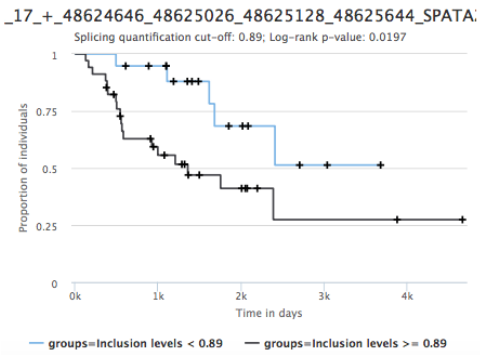


Figure 5.12: Survival curves for a cut-off of 99% of exon inclusion for the splicing event *SE 17 + 48624646 48625026 48625128 48625644 SPATA20*.

We may hypothesise that the promotion of isoforms including the exon of the *SE 17 + 48624646 48625026 48625128 48625644 SPATA20* event in patients with adrenocortical carcinoma may lead to a smaller lifespan, as the 5-year survival rate (i.e. the number of patients alive after 5 years) is 41% for patients whose samples measured exon inclusion levels of 89% or higher versus 68% for the remaining patients.

Finally, the associated gene, transcript and protein annotation is also available to explore the isoforms and respective proteins that may be affected by the alternative splicing event of interest (figure B.7). This section of the program also includes relevant literature and interestingly, although the gene SPATA20 is not reported to be associated to adrenocortical carcinoma, it is a biomarker of bile duct cancer [107].

Chapter 6

Testing

6.1 Unit and Continuous Testing

Unit tests are automated and reproducible executions that validate if a unit of code returns an expected value, warning the developer if any existing functionality was broken after changing the code [98]. In R, unit testing can be incorporated using the R package testthat [45]. Each test file comprises an arbitrary unit of code which can range from testing a single function to a collection of related functions [45].

Similarly, CI automatically builds and tests projects on every change to a repository to ensure the program is still working after each modification [108]. One exemplary tool is Travis-CI, a remotely hosted CI that works exclusively with GitHub[1]. Travis-CI supports a lot of languages including R. Travis-CI was added to the project to check if the package passes most requirements of Bioconductor in a Linux virtual machine, making it easy to know when something breaks. It tests against CRAN and Bioconductor running the commands `R CMD build` and `R CMD check`, which need to be passed with no errors or warnings for the package to be accepted in Bioconductor. The command `R CMD BiocCheck` must also be tested afterwards but it is not part of Travis-CI.

Another CI service used is AppVeyor. Similar to Travis-CI, AppVeyor builds and tests the package in a Windows virtual machine and works as a complement of Travis-CI.

Before Travis-CI finishes running, it asks CodeCov to test the code coverage of the project. This allows to know the percentage of lines in the code covered by unit tests. Unfortunately, functions with objects from Shiny are not testable. An obvious workaround is to create normal functions that receive Shiny objects as arguments.

6.2 Usability Testing

Usability testing evaluates the interface of a program and its quality by identifying design issues. Improving the problems encountered between different test sessions also allows to check if the potential fixes resolved pending issues. Ultimately, usability testing may improve user satisfaction [109, 110].

The laboratory usability testing is a common type of usability testing where typical tests are performed by testers following a detailed session script in a controlled environment to identify interface design issues such as the number, type and severity of errors [111].

[1] `https://travis-ci.com` (last accessed on 20 September 2016)

Another common type of usability testing is known as condensed contextual inquiry where the participants are observed in their workplace or home while performing the activities without a strict guide, taking much more time than other types of usability testing [111]. However, it allows to gather more data by maintaining an ongoing dialogue with the test participants without influencing their responses.

The laboratory testing can also be conducted in the environment and computers of the participants in a hybrid approach called field usability testing [111], where test subjects are asked to perform higher-level tasks and the interaction may be less scripted.

After discussing the nature of the tests with my advisors, we agreed that the usability test script should be divided into two parts: the first is a rigorous and detailed script with defined tasks, while the second part is more flexible with only a higher-level task in order to allow test subjects to discuss more freely what they think of the program's interface and other attributes (read section C.1: Script).

The selected test participants are 6 members of the Computational Biology Lab from Instituto de Medicina Molecular and they represent the target audience: users with an expertise on the domain knowledge (alternative splicing quantification and analysis). The downside of selecting these participants is their familiarity with the project.

The tests were performed in the machine server used by the Computational Biology lab with which all members are familiar and located in the workplace of the group. All test sessions shared the same script that encouraged thinking aloud. Interaction with the test participants was kept to a minimum during task completion, although the last question promotes interactivity with the test subjects. In concordance with previous studies [109–111], the following metrics were measured while performing the tests:

- Time to complete a task (in seconds)
- Number of usability problems encountered for each task (design issues)
- Comments made while performing the activities
- Opinion regarding the completed task
- Satisfaction regarding the interface used to complete the task based on a Likert-scale (not satisfied, satisfied, good, very good and excellent) [110]

Note that all measurements were recorded by a single person, which made timing the activities (a task suggested to be performed by a log-keeper [110]) less accurate. Also, for the same reason, not all tasks were timed.

After each test, the program was improved to tackle some of the issues encountered by the tester. Thus, each test underwent a different version of the program, aiming to correct some of the previous tester's complaints and implement their suggestions[2].

One of the tasks asked to test participants was to describe the program functionality from the welcome message available. The answers were pretty similar: the program loads, quantifies and analyses transcriptomic data from TCGA or from files in the computer. The aim of the program seems to be clear by reading the welcome message.

There were around 45 distinct design issues noted by the test participants of which 25 (56%) were solved (see figure C.1). One glaring issue of the program was that the users spent a lot of time (up to 6

[2]The change log between the different versions of the program used in usability tests are available at `https://github.com/nuno-agostinho/psichomics/releases`.

minutes) looking for the survival analyses. Although users understood they should click on the *Analyses* tab, they did not notice the select menu where they could change the current analysis. To decrease time spent here, a dropdown containing a list of analyses is now accessible by simply clicking on the *Analyses* tab. Data grouping was also confusing and two testers took about 7 minutes to understand how it worked. This was improved by changing the interface and adding contextual clues (disable group selection when no group is available, animate button to create groups when user clicks on a disabled group selection input, etc).

The satisfaction for the different interfaces had very good/excellent scores on average for each task (figure 6.1). The worst score is an average of 4 and corresponds to the survival analysis. One of the reasons for the sub-optimal score may be the testers' lack of experience with the interface for data grouping. Besides, some of the test participants are not experienced in performing survival analysis either.

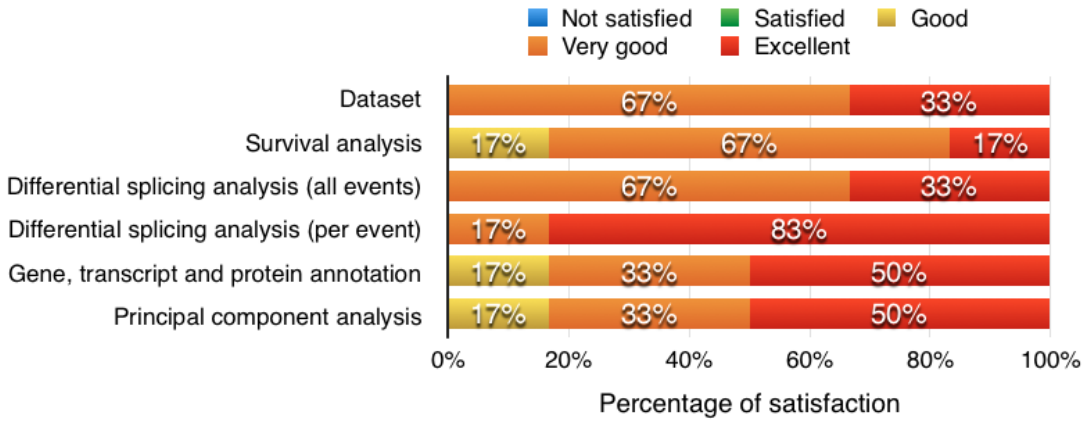


Figure 6.1: Percentage of satisfaction of all test participants based on a Likert scale for the different interfaces available in the program.

The test participants were constantly suggesting enhancements as well, including: add comparison between transcripts and proteins, link protein domains to respective UniProt page, rewrite messages to be more precise, etc. Some of these suggestions were implemented, while others suggestions are being considered.

Usability testing was conducted in a late stage of the program development. Although the members of the Computational Biology group were present while developing the program, there is a striking difference between the program before and after the test sessions, as already mentioned above regarding some of the implemented improvements.

Finally, to further improve usability, external colleagues with expertise in analysing alternative splicing were asked to examine and give feedback on the program by following the steps of a user tutorial regarding the visual interface and their feedback is expected soon.

6.3 Benchmarking

The most time-consuming steps of the program were worked on to take the least possible amount of time. These steps comprise data loading, alternative splicing quantification and differential splicing

analysis.

Common tumour types from TCGA were selected for this benchmark, including the liver hepatocellular carcinoma data of 377 patients (a close number to the average and median number of available patients per tumour type) and the breast invasive carcinoma data of 1097 patients (the tumour type data containing the most patients in TCGA).

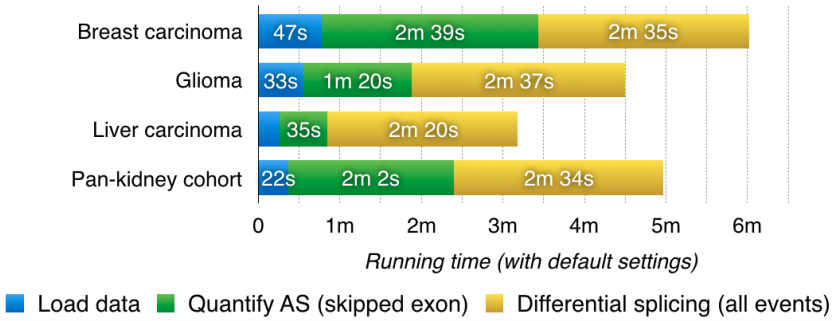


Figure 6.2: Processes' running time average over 10 runs for multiple tumour types. The tests were performed using the visual interface.

The time to load data, quantify alternative splicing and analyse differential splicing of all splicing events is largely dependent on the number of samples available. The performance of differential splicing analysis is also dependent on the number of groups that are compared and the type and number of statistical analyses performed.

The visual interface adds an overhead to the performance. To account for the worst-case scenario, the benchmarking was performed in the visual interface.

Chapter 7

Deployment

The project is publicly hosted online in GitHub (`https://github.com/nuno-agostinho/psichomics`) and is currently under review to be accepted in Bioconductor [17]. These two modes of distribution allow the program to be installed and locally run on the user's computer. However, it would be more appropriate for the application to be remotely accessible through a web browser. Therefore, there are plans to deploy the application in a web server following the ongoing upgrade in the computational infrastructure available at Instituto de Medicina Molecular.

Nevertheless, the application still needs modifications before being hosted in a web server. At the moment, file loading is based on looking for valid files in a given folder. Unfortunately, this can only be done by R in a local installation because HTML5 does not allow to prompt for folders (only for one or multiple files)[1]. A workaround would be to ask the user to zip the folder of interest, although that idea departs from good usability.

As the tool is currently available in GitHub, the *master* and the *dev* branches are used to distribute a stable and a development version of the package, respectively. Whenever the changes accumulated in the development branch are satisfactory and pass the CI tools without issues, they are merged with the stable branch. A new version release may even be accompanied with a change log.

7.1 License

An open-source program requires a suitable license. Licenses can be permissive enough to allow developers to modify and distribute the program and use it commercially and privately, with the condition that a copy of the license and copyright notice are included in the code (e.g. MIT license and Apache License 2.0). There are also copyleft licenses such as Mozilla Public License 2.0 and derivatives of the GNU GPLv3 licenses that demand the use of the original license in any work derivatives and to disclose the source code (depending on the license, the source code to be disclosed may be the modifications performed to the GPL-licensed work or even the full source code of the program).

The current public version of the software holds an MIT license and all original or modified code from external sources is compliant with the chosen license, including the external JavaScript and CSS libraries distributed with the program.

[1] `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input` (last accessed on 20 September 2016)

Chapter 8

Conclusions

With the increasing amount and resolution of biological data available, we are able to study biological processes with more precision than ever before [1]. There is a great amount of data available online in public repositories that can be freely used to analyse processes of interest, such as the data from TCGA [10]. These data can be used to study biological process such as alternative splicing, which is a major contributor to protein diversity in many organisms [2,3]. Reportedly, the deregulation of alternative splicing may foster the progression of cancer and other diseases [4–8].

PSIchomics is presented as a tool to quantify, analyse and visualise alternative splicing in tumour and normal samples from TCGA with the assistance of interactive visualisations and analyses, including survival, principal component and differential splicing analysis. It tries to fill the gap from other analytical tools currently used by:

- Quantifying alternative splicing from already processed data (read section 3.3: Alternative Splicing Quantification)
- Incorporating available clinical information.
- Performing analysis of variance.
- Possessing an easy-to-use interface (read section 6.2: Usability Testing).

To accommodate the rapidly changing field of bioinformatics, the program's architecture was designed around modifiability to promote modularity, extensibility, unit testing and collaboration with developers and biologists interested in analysing alternative splicing (read section 4.4: Architecture). Templates and examples are provided in the program to allow developers to create and extend the program with support for new databases and file formats, new analyses and visualisations or new data processing tools.

A full-fledged analysis (including clinical information and junction quantification data loading, alternative splicing quantification and differential splicing analysis) can take less than 10 minutes for more than 1000 patients (see section 6.3: Benchmarking). As exemplified in section 5.5: Case Study, users of this program can find alternative splicing signatures that are specific of a tumour type, tumour stage or other clinical attributes. These splicing events can then be further studied to validate them as putative prognostic factors and even therapeutic targets.

The application is publicly available in GitHub (`http://github.com/nuno-agostinho/psichomics`) and is under review in Bioconductor to be released by Mid-October 2016.

8.1 Future Work

GitHub has a feature to list software bugs, suggestions and improvements. That feature (called *GitHub issues*) has been used to list all future improvements and questions, some of which are mentioned in this document[1], such as the implementation of beta regression analysis to study the distribution of PSI values (read subsection 3.4.3: Differential Splicing Analysis).

The performance of the program can also be improved. For instance, sections of time-consuming functions could be converted to C++ for performance improvements[2] and/or parallelised (specially when performing multiple independent calculations, such as in the case of differential analyses).

Although the package quantifies and analyses alternative splicing data using TCGA data, it could have a broader appeal by accepting data from other databases. For example, a few members of the Computation Biology lab at Instituto de Medicina Molecular analyse alternative splicing using data from neurodegenerative diseases and normal tissue. Consequently, an ambitious goal of the application would be to study alternative splicing changes outside of the context of cancer by including, for instance, GTEx support. Unfortunately, data from this project is currently unaccessible given that they do not share a public API.

We are also awaiting the upgrade of the computer infrastructure at Instituto de Medicina Molecular to deploy the program in a web server. Allowing users to remotely access the program through a web browser has two major advantages: it avoids the installation of all the associated software and allows to always use the latest version of the program available.

[1] `https://github.com/nuno-agostinho/psichomics/issues`

[2] Note that third-party C functions are already employed in some cases, such as when loading files.

Appendix A

Source code

A.1 Format of the Clinical Information

```
firehoseClinicalFormat <- function() {
    list(
        tablename   = "Clinical_data",
        filename    = "clin.merged.txt",
        description = "Clinical_data_of_the_patients",
        dataType    = "Clinical_data", # General data category

        # Transpose the data? This is the first step before
        # parsing the information. After transposition, a row
        # of the current data equals a column of the original.
        transpose   = TRUE,

        # Format checker information
        rowCheck    = TRUE,  # Check format by row or column
        checkIndex  = 1,     # Row/column index to check the format

        # File string to check
        check = c("admin.batch_number", "admin.bcr",
                  "admin.day_of_dcc_upload",
                  "admin.disease_code", "admin.file_uuid",
                  "admin.month_of_dcc_upload"),

        # Parsing information
        delim    = "\t",  # Delimiter used to separate fields
        colNames = 1,     # Row to use for column names
        rowNames = "patient.bcr_patient_barcode", # Column for row names
        ignoreCols  = NULL,  # Columns to ignore
        ignoreRows  = 1:2,   # Rows to ignore
        commentChar = NULL,  # Comment symbol; comments are ignored
```

```
        # Other options
        unique = FALSE,    # Remove duplicated rows

        # Default columns to show (NULL to show all)
        show = c("patient.stage_event.pathologic_stage_tumor_stage",
                 "patient.vital_status", "patient.race",
                 "patient.days_to_death",
                 "patient.days_to_last_followup",
                 "patient.radiation_therapy", "patient.gender",
                 "patient.clinical_cqcf.histological_type",
                 "patient.ethnicity", "patient.race_list.race"),

        process = function(data) {
            # Add suffix to identify column detailing tumour stages
            col <- grep("stage.*pathologic_stage", colnames(data))
            colnames(data)[col] <- paste0(colnames(data)[col],
                                          "_tumor_stage")
            return(data)
        }
    )
}

attr(firehoseClinicalFormat, "loader") <- "formats"
```

A.2 Template for an Analysis File

```
#' User interface of template
#' @param id Character: namespace identifier
#'
#' @importFrom shiny NS tagList sidebarPanel mainPanel sliderInput
#' actionButton uiOutput
#' @importFrom highcharter highchartOutput
#'
#' @return HTML elements for the interface of the template
templateUI <- function(id) {
    ns <- NS(id) # Identifier
    tagList(
        sidebarPanel( # Sidebar interface
            sliderInput(inputId=ns("num"), label="Number␣of␣samples",
                        min=1, max=20, value=4),
            actionButton(inputId=ns("submit"), label="Plot")),
        mainPanel( # Main interface
            uiOutput(outputId=ns("text")),
            highchartOutput(outputId=ns("plot"))))
}
```

```
#' Server logic of template
#'
#' @param input Shiny input
#' @param output Shiny output
#' @param session Shiny session
#'
#' @importFrom shiny renderUI observeEvent isolate tagList tags
#' @importFrom highcharter renderHighchart
templateServer <- function(input, output, session) {
    # Wait for user to press the button
    observeEvent(input$submit, {
        # Break reactive dependence
        num <- isolate(input$num)

        # Retrieve random samples
        sample <- sort( sample(100, size=num, replace=TRUE) )

        # Print samples
        output$text <- renderUI(tagList(tags$b("Samples:"),
                                        paste(sample, collapse="␣")))
        # Plot frequency
        output$plot <- renderHighchart( hchart(sample) )
    })
}

attr(templateUI, "loader") <- "analysis"
attr(templateUI, "name") <- "Template"
attr(templateServer, "loader") <- "analysis"
```

Appendix B

Screenshots

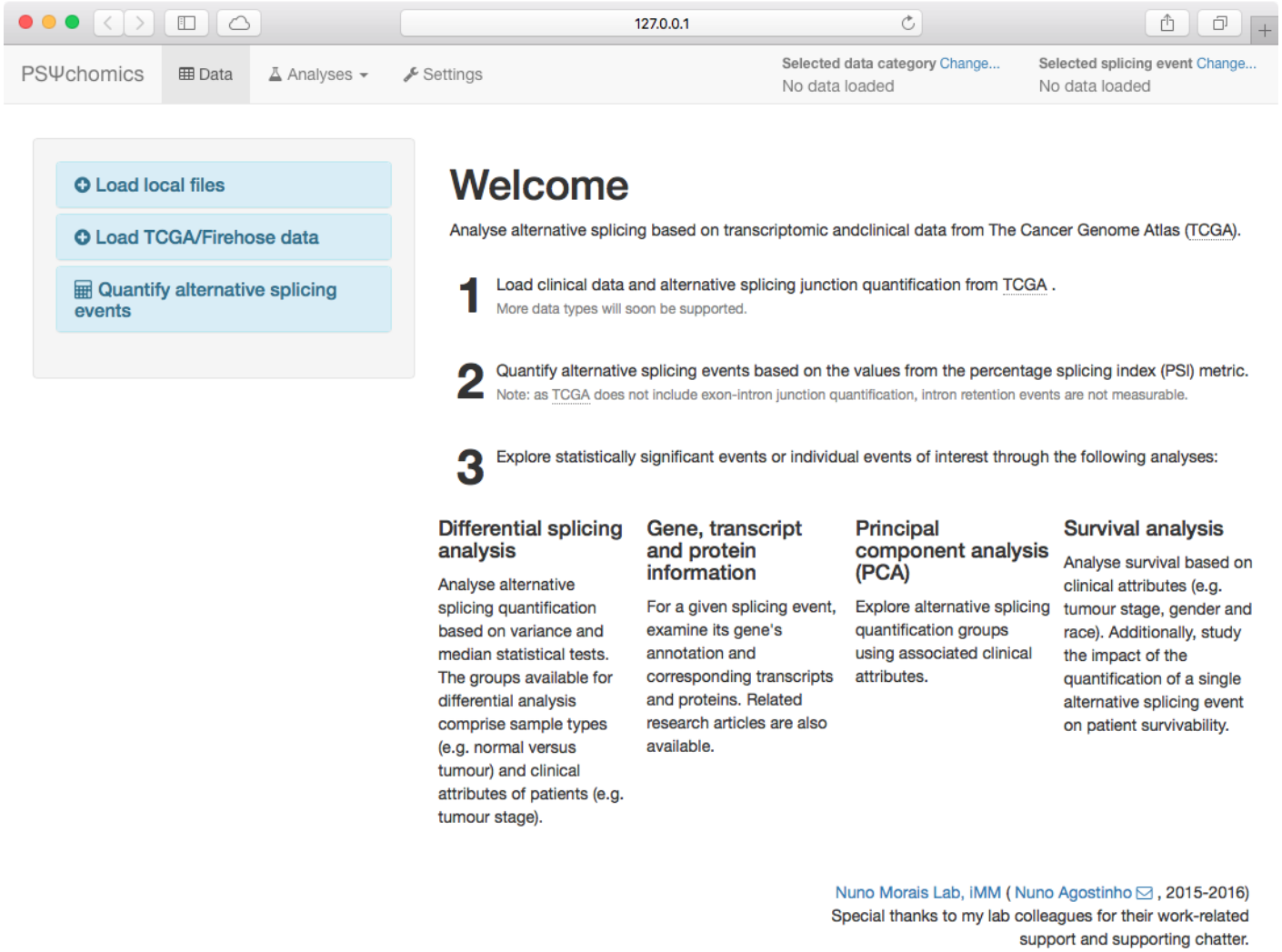


Figure B.1: The welcome message presents instructions to the users when the visual interface is first loaded and while it has not loaded any data. The instructions aim to be clear and concise to explain the program's functionality to the user.

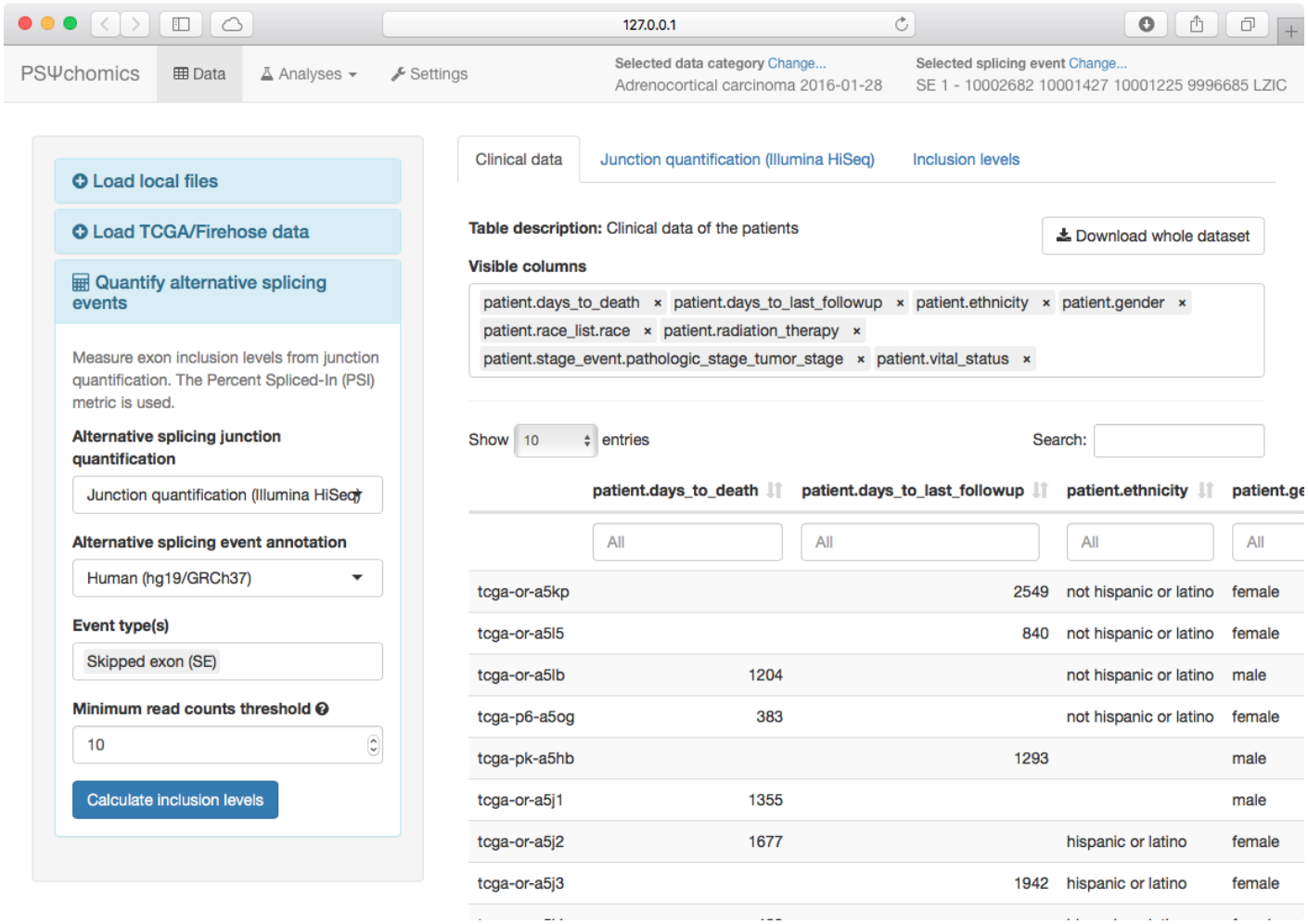


Figure B.2: Dataset interface with loaded data relative to samples of patients with adrenocortical carcinoma.

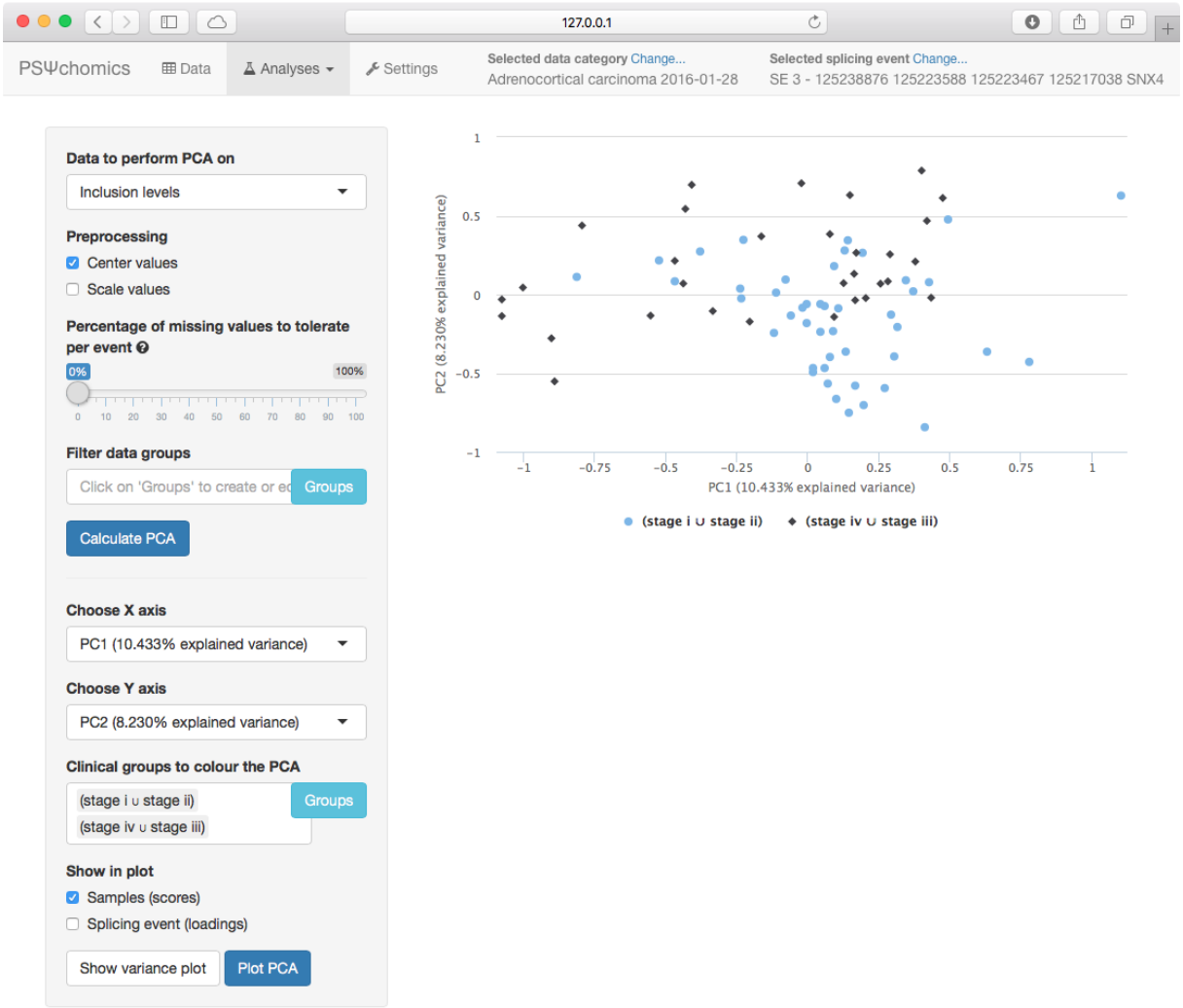


Figure B.3: Interface for principal component analysis (PCA). The plot depicts PCA of the alternative splicing quantification for samples of patients with adrenocortical carcinoma.

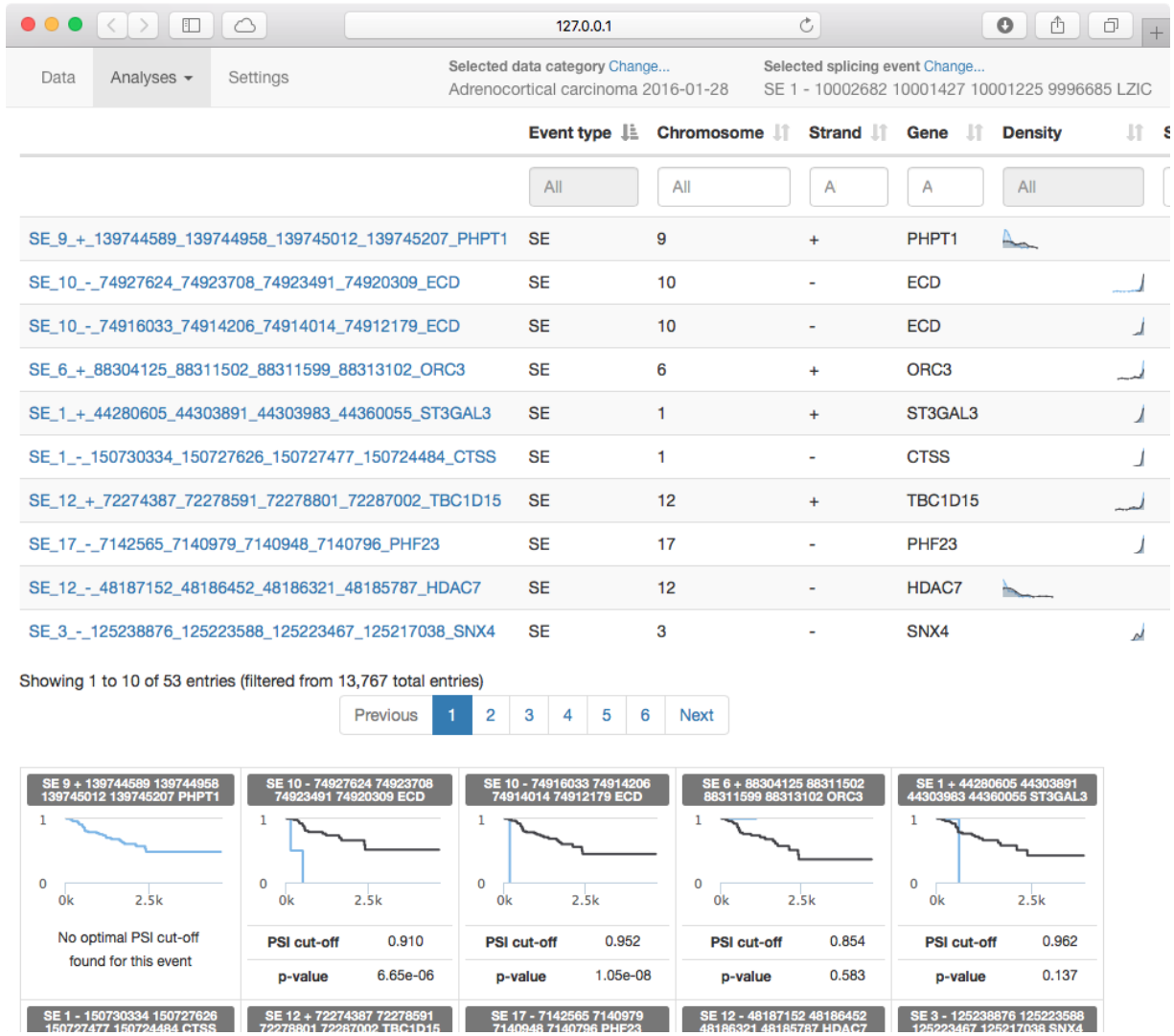


Figure B.4: Differential splicing analysis based on tumour stages from samples of patients with adrenocortical carcinoma and survival analysis based on the PSI cut-off that maximises the significance of difference in survival between individuals separated by that threshold for the selected event. The table is filtrable and sortable. Clicking on an event in table takes the user to a page with the statistical analysis for that event, while clicking on the survival plot takes the user to a page dedicated to survival analysis.

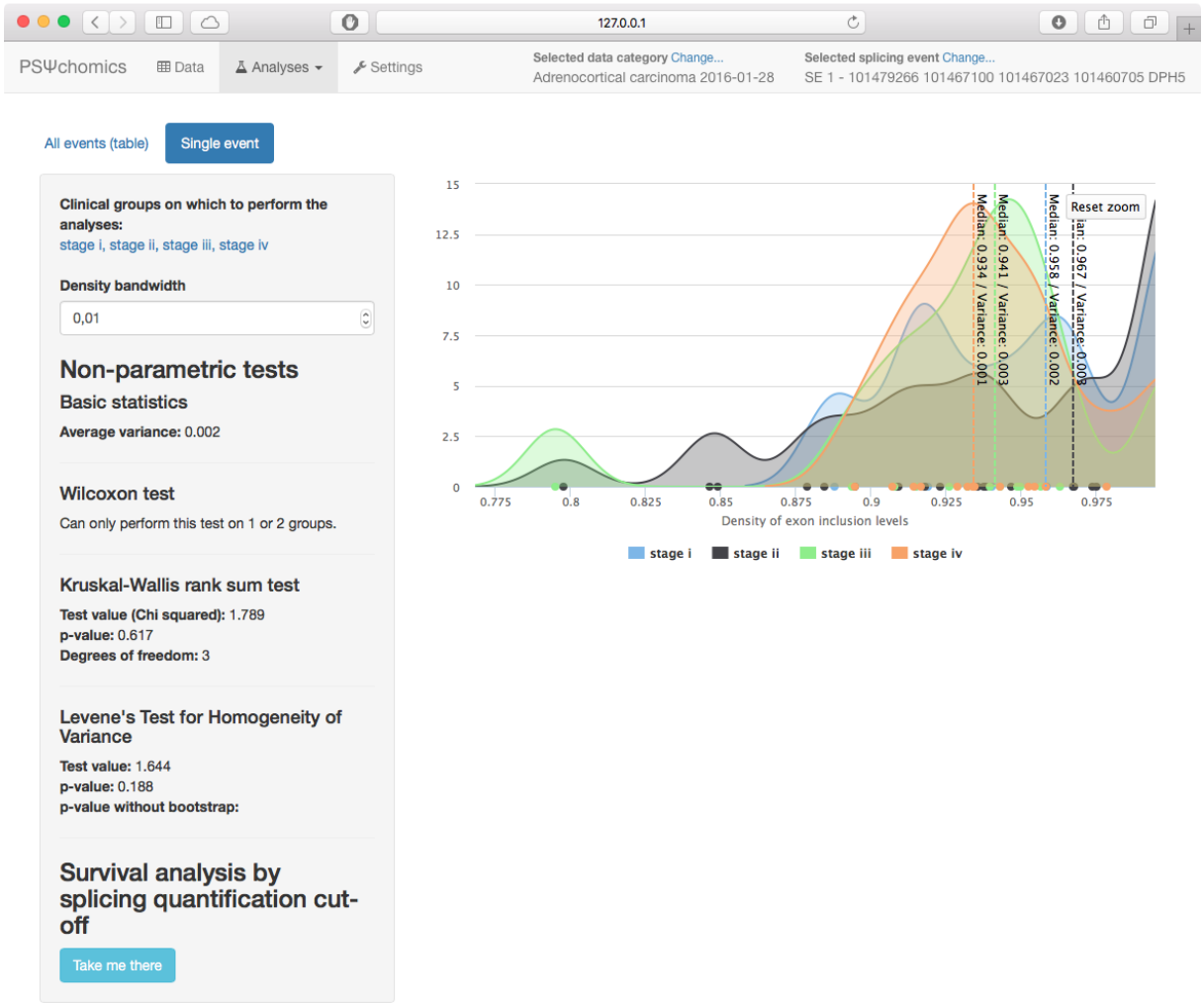


Figure B.5: Interface of differential splicing analysis for one event showing the distribution of alternative splicing quantification for samples of patients with adrenocortical carcinoma across tumour stages.

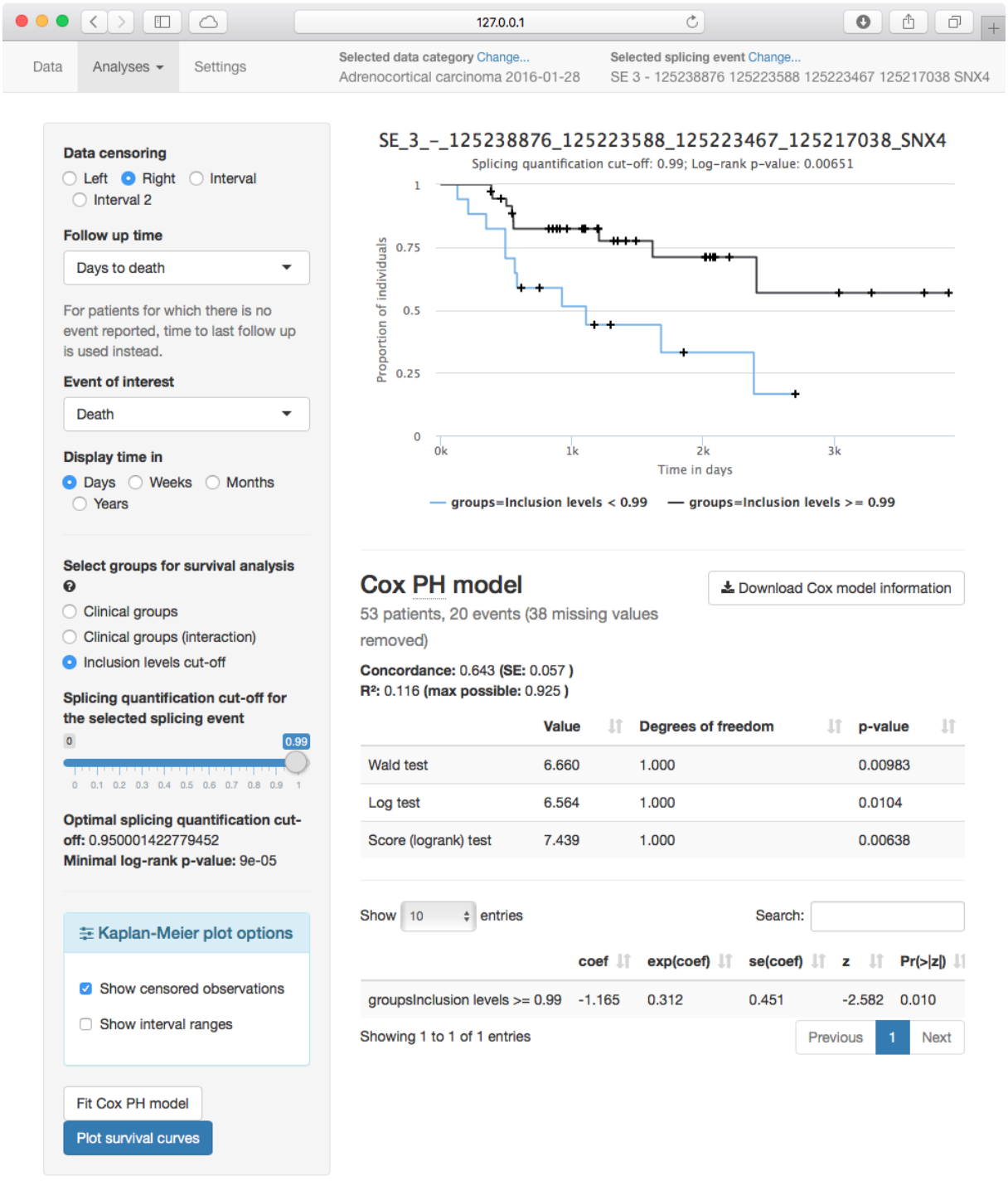
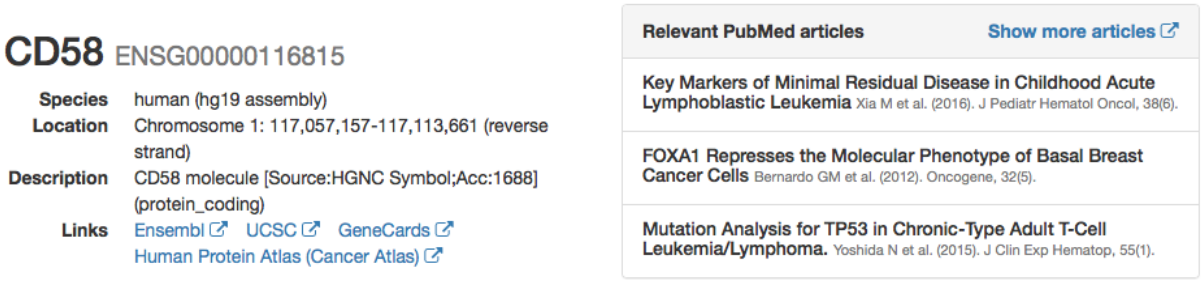
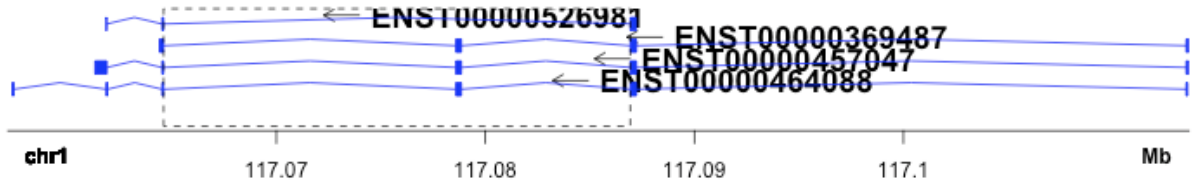


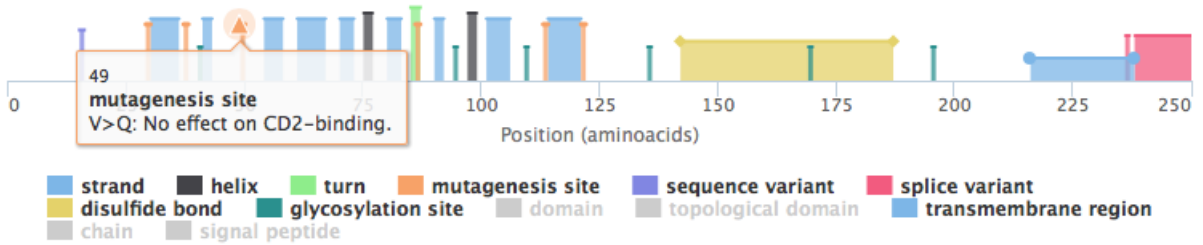
Figure B.6: Survival curves (top) and Cox proportional hazards model fit (bottom) for patients with adrenocortical carcinoma separated by a PSI cut-off for a given alternative splicing event.



(a) Gene information including links to external databases and relevant literature.



(b) Static plot of the gene's transcripts highlighting an alternative splicing (skipped exon) event.



(c) Interactive plot of a selected UniProt protein with some protein features omitted (light gray) and showing a tooltip for a specific feature.

Figure B.7: Gene, transcripts and protein annotation provided by PubMed Central (a), Ensembl (a and b) and UniProt (c) for a given gene.

Appendix C

Usability Testing

C.1 Script

This script was used during the sessions with test participants (see section 6.2: Usability Testing).

C.1.1 Instructions

I will present you a new tool to analyse alternative splicing from tumour transcriptomic data. The tool allows you to load data from The Cancer Genome Atlas (TCGA) database and to perform exploratory and survival analyses, as well as differential splicing analysis using non-parametric statistical tests.

This enquiry evaluates the tool's interface and user interaction to assess a user can easily and quickly analyse data. I will give you general tasks and pose some questions and let you explore the program to give me the answers. The time you take to answer will be recorded to compare with other testers, in order to identify common time-consuming steps.

At any time, you may ask questions regarding the tool, but please understand I will avoid giving you the answer right away unless you are unable to proceed.

Finally, I would like to stress the importance of thinking aloud. As the tests serve the purpose of analysing the tool's intuitiveness and usability, I would like to know your train of thought when performing the tasks to understand what catches your attention and what may confuse you. I would also like to know of any suggestions you may have regarding how you complete the tasks.

If you have any further questions, I will be happy to answer them. Else, we may now start the test.

C.1.2 General Questions

1. Is the interface clear and intuitive?

2. How do you describe your satisfaction with the interface regarding the completed tasks: not satisfied, satisfied, good, very good or excellent?

C.1.3 Tasks

1 Read and follow the instructions from the GitHub page to start the program.

1.1 Describe the interface. What do you think the software allows you to do?

2 Load the most recent version of Bladder Urothelial Carcinoma (BLCA) data from TCGA (both clinical data and junction quantification).

2.1 Is the message that appears clear to you? Why do you click the "Load data" button twice?[1]

2.2 How many patients are loaded? How many are female? How many are asian females? How many patients in total are in stage III of the disease?

3 Analyse survival data.

3.1 Analyse survival data by tumour stage of the disease with time to death as the follow-up time and death as the event of interest. Perform a Kaplan-Meier plot and fit a Cox proportional hazards model.

3.1.1 Are there groups with zero survivors? Are the curves' differences significant?

3.2 Analyse survival data by tumour stage, gender and race. Perform a Kaplan-Meier plot and fit a Cox proportional hazards model.

3.2.1 Are there groups with zero survivors? Are the curves' differences significant?

3.3 Is group creation clear? What is the difference between group selection types?

3.4 Try zooming in any point of interest. Try omitting some data series.

4 Analyse differential splicing using all the statistical tests.

4.1 Is the error message "missing splicing events quantification" clear? Load missing data.[2]

4.2 Filter the statistical significant events and sort them by variance difference.

4.3 Include survival curves separated by the optimal alternative splicing quantification cut-off of each event (time to death and death events).

5 Choose a relevant alternative splicing event from the top of the previous table.

5.1 Do all groups share the same distribution of alt. splicing quantification? What is the median and variance of the primary solid tumour group?

5.2 Using the selected event, plot a Kaplan-Meier that maximises the survival difference for a given cut-off of the alternative splicing quantification. Is the difference significant?

5.3 Which gene, transcripts and proteins are related to the selected alternative splicing event?

5.4 Check if there's any evidence of the association of that gene with cancer in the literature.

6 Perform principal component analysis (PCA) on the alternative splicing quantification and colour by tumour stage.

7 Find relevant alternative splicing events for another type of tumour.

[1] A modal message appears to inform that the data is being downloaded and the user needs to notify the program when the download has finished.

[2] This error message appears if the user did not yet quantify splicing.

C.2 List of Design Issues Encountered

I1. Problem opening visual guide

I2. Indicate most recent sample

I3. *Folder to store data* is not clear

I4. Improve text of the *downloading data* message

I5. Filtering in *Data* tab is exclusively for data visualising purposes

I6. Suggest unique values when filtering a data column

I7. Misunderstood message that required data is not available

I8. Problem finding number of loaded patients

I9. Download the whole subset of a table

I10. Make it easier to change analyses

I11. Calculate optimal sample size for survival curves to avoid having curves with a small number of patients

I12. Show number of patients in each survival curve

I13. Tries to write name of groups of interest; does not click in *Groups* button to create groups

I14. Did not find the column for tumour stages

I15. Tried selecting groups with the mouse (click and drag)

I16. User did not understood groups were available after clicking *create group*

I17. Tries to create intersecting groups from the group selection instead of using a formula

I18. Formula interface has no instructions

I19. Zooming and series omitting in interactive plots are not intuitive

I20. Not clear how to filter columns

I21. *Load junction quantification* message not understood

I22. Confused by the *uncheckable* statistical tests

I23. Rewrite message regarding *too many missing values*

I24. No indication that alternative splicing quantification is finished

I25. Use clinical groups in differential splicing analysis

I26. Restrict analysis to paired-matched samples

I27. Manual column filtering of tables is inflexible

I28. Improve column names of the differential splicing analysis table

I29. Does not notice options for survival analyses in differential analysis

I30. Table is redrawn when calculating optimal PSI values to separate survival curves; custom state of the table is lost

I31. Finds text for PSI cut-off confusing

I32. Not sure how to get to differential analysis per splicing event

I33. Allow to click in density plot to go to differential analysis per splicing event

I34. Testers do not notice event of interest is selected

I35. Confusing when taking user to survival curves after a plot was drawn

I36. Protein selection is not intuitive

I37. Improve display of the transcript names and better highlight the splicing event

I38. No indication of the plots correspond to transcripts

I39. Add relevant PubMed articles

I40. Explain difference between the two group selection boxes in PCA

I41. Add information on value imputation

I42. Create groups based on events or individuals of interest

I43. Inform PCA calculation has finished

I44. Tester does not remember where to separate survival curves by alternative splicing quantification

I45. Message asking to *replace data* appears even if data is going to be downloaded

C.3 Table of Design Issues by Task and Tester

Task	Issue	U1	U2	U3	U4	U5	U6
1	I1	Problem					
1.1	I2	Try		Solved			
1.1	I3	Problem	Problem		Problem		Problem
1.1	I4	Problem		Try	Solved		
2.1	I5	Problem					
2.1	I6	Problem	Problem	Problem	Solved		
2.1	I7		Solved				
2.2	I8			Problem			

2.2	I9			Solved			
3	I10	Problem	Try	Problem	Problem	Solved	
3	I11	Problem					
3	I12	Solved					
3.1	I13	Problem	Try	Problem	Problem	Solved	
3.1	I14		Problem	Problem	Problem	Try	Solved
3.1	I15		Problem	Problem		Problem	
3.1	I16	Try			Solved		
3.2	I17	Problem	Solved				
3.2	I18	Solved					
3.4	I19			Problem	Problem	Problem	Problem
4	I20			Problem			
4.1	I21		Problem	Try	Solved		
4.1	I22		Problem				
4.1	I23				Solved		
4.1	I24		Solved				
4.2	I25					Problem	Solved
4.2	I26					Problem	
4.2	I27					Problem	Problem
4.2	I28					Solved	
4.3	I29		Problem				
4.3	I30	Problem	Problem	Problem	Problem	Problem	Solved
4.3	I31			Problem			Problem
5.1	I32	Solved					
5.1	I33	Problem	Solved				
5.2	I34	Problem	Problem	Problem	Solved		
5.2	I35			Problem	Problem	Solved	
5.2	I36					Problem	
5.3	I37	Problem					
5.3	I38		Solved				
5.3	I39			Problem	Solved		
6	I40	Try			Problem		Problem
6	I41	Solved					
6	I42		Problem				
6	I43					Problem	
7	I44		Problem				
7	I45			Problem			

Table C.1: Design issues by task and tester encountered during usability testing. The list of issues is available in section C.2: List of Design Issues Encountered. Legend: *problem* is an issue that was not solved; *try* is an issue that was not successfully solved; and *solved* refers to an issue that was taken care of. An issue is considered solved unless the next test participants encounter a similar design issue.

Bibliography

[1] Zhong Wang, Mark Gerstein, and Michael Snyder. RNA-Seq: a revolutionary tool for transcriptomics. *Nature reviews. Genetics*, 10(1):57–63, January 2009.

[2] Eric T Wang, Rickard Sandberg, Shujun Luo, Irina Khrebtukova, Lu Zhang, Christine Mayr, Stephen F Kingsmore, Gary P Schroth, and Christopher B Burge. Alternative isoform regulation in human tissue transcriptomes. *Nature*, 456(7221):470–476, November 2008.

[3] Yeon Lee and Donald C Rio. Mechanisms and Regulation of Alternative Pre-mRNA Splicing. *Annual review of biochemistry*, 84(1):291–323, June 2015.

[4] S Oltean and D O Bates. Hallmarks of alternative splicing in cancer. *Oncogene*, 33(46):5311–5318, November 2014.

[5] Anita Sveen, Bjarne Johannessen, Manuel R Teixeira, Ragnhild A Lothe, and Rolf I Skotheim. Transcriptome instability as a molecular pan-cancer characteristic of carcinomas. *BMC genomics*, 15(1):672, January 2014.

[6] Heidi Dvinge and Robert K Bradley. Widespread intron retention diversifies most cancer transcriptomes. *Genome medicine*, 7(1):45, 2015.

[7] Endre Sebestyén, Babita Singh, Belén Miñana, Amadís Pagès, Francesca Mateo, Miguel Angel Pujana, Juan Valcárcel, and Eduardo Eyras. Large-scale analysis of genome and transcriptome alterations in multiple tumors unveils novel cancer-relevant splicing networks. *Genome research*, 26(6):732–744, June 2016.

[8] Benoit Chabot and Lulzim Shkreta. Defective control of pre-messenger RNA splicing in human disease. *The Journal of Cell Biology*, 212(1):13–27, January 2016.

[9] Manuel Irimia, Robert J Weatheritt, Jonathan D Ellis, Neelroop N Parikshak, Thomas Gonatopoulos-Pournatzis, Mariana Babor, Mathieu Quesnel-Vallières, Javier Tapial, Bushra Raj, Dave O'Hanlon, Miriam Barrios-Rodiles, Michael J E Sternberg, Sabine P Cordes, Frederick P Roth, Jeffrey L Wrana, Daniel H Geschwind, and Benjamin J Blencowe. A highly conserved program of neuronal microexons is misregulated in autistic brains. *Cell*, 159(7):1511–1523, December 2014.

[10] Katarzyna Tomczak, Patrycja Czerwińska, and Maciej Wiznerowicz. The Cancer Genome Atlas (TCGA): an immeasurable source of knowledge. *Contemporary oncology (Poznań, Poland)*, 19(1A):A68–77, 2015.

[11] Dorothea Emig, Nathan Salomonis, Jan Baumbach, Thomas Lengauer, Bruce R Conklin, and Mario Albrecht. AltAnalyze and DomainGraph: analyzing and visualizing exon expression data. *Nucleic Acids Research*, 38(Web Server issue):W755–62, July 2010.

[12] Michael Ryan, Wing Chung Wong, Robert Brown, Rehan Akbani, Xiaoping Su, Bradley Broom, James Melott, and John Weinstein. TCGASpliceSeq a compendium of alternative mRNA splicing in cancer. *Nucleic Acids Research*, 44(D1):D1018–22, January 2016.

[13] Zhenzhen Huang, Huilong Duan, and Haomin Li. TCGA4U: A Web-Based Genomic Analysis Platform To Explore And Mine TCGA Genomic Data For Translational Research. *Studies in health technology and informatics*, 216:658–662, 2015.

[14] Yarden Katz, Eric T Wang, Edoardo M Airoldi, and Christopher B Burge. Analysis and design of RNA sequencing experiments for identifying isoform regulation. *Nature methods*, 7(12):1009–1015, December 2010.

[15] Shihao Shen, Juw Won Park, Zhi-xiang Lu, Lan Lin, Michael D Henry, Ying Nian Wu, Qing Zhou, and Yi Xing. rMATS: Robust and flexible detection of differential alternative splicing from replicate RNA-Seq data. *Proceedings of the National Academy of Sciences*, 111(51):E5593–E5601, 2014.

[16] Gael P Alamancos, Amadís Pagès, Juan L Trincado, Nicolás Bellora, and Eduardo Eyras. Leveraging transcript quantification for fast computation of alternative splicing profiles. *RNA*, 21(9):1521–1531, September 2015.

[17] Robert C Gentleman, Vincent J Carey, Douglas M Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean Y H Yang, and Jianhua Zhang. Bioconductor: open software development for computational biology and bioinformatics. *Genome biology*, 5(10):R80, 2004.

[18] Joshua Kunst. *highcharter: A Wrapper for the 'Highcharts' Library*, 2016. R package version 0.3.0.

[19] Winston Chang, Joe Cheng, J J Allaire, Yihui Xie, and Jonathan McPherson. *shiny: Web Application Framework for R*, 2015.

[20] Mario Deng. *FirebrowseR: An R client for broads firehose pipeline, providing TCGA data sets*, 2016.

[21] Binsheng Gong, Charles Wang, Zhenqiang Su, Huixiao Hong, Jean Thierry-Mieg, Danielle Thierry-Mieg, Leming Shi, Scott S Auerbach, Weida Tong, and Joshua Xu. Transcriptomic profiling of rat liver samples in a comprehensive study design by RNA-Seq. *Scientific data*, 1:140021, 2014.

[22] Francis Crick. Central dogma of molecular biology. *Nature*, 227(5258):561–563, August 1970.

[23] E Peter Geiduschek and Robert Haselkorn. Messenger RNA. *Annual review of biochemistry*, 38(1):647–676, 1969.

[24] Mark B Gerstein, Can Bruce, Joel S Rozowsky, Deyou Zheng, Jiang Du, Jan O Korbel, Olof Emanuelsson, Zhengdong D Zhang, Sherman Weissman, and Michael Snyder. What is a gene, post-ENCODE? History and updated definition. *Genome research*, 17(6):669–681, June 2007.

[25] Phillip A Sharp. The discovery of split genes and RNA splicing. *Trends in biochemical sciences*, 30(6):279–281, June 2005.

[26] Analisa DiFeo, John A Martignetti, and Goutham Narla. The role of KLF6 and its splice variants in cancer therapy. *Drug resistance updates : reviews and commentaries in antimicrobial and anticancer chemotherapy*, 12(1-2):1–7, February 2009.

[27] Heebal Kim, Robert Klein, Jacek Majewski, and Jurg Ott. Estimating rates of alternative splicing in mammals and invertebrates. *Nature genetics*, 36(9):915–916; author reply 916–917, September 2004.

[28] Nuno L Barbosa-Morais, Manuel Irimia, Qun Pan, Hui Y Xiong, Serge Gueroussov, Leo J Lee, Valentina Slobodeniuc, Claudia Kutter, Stephen Watt, Recep Colak, TaeHyung Kim, Christine M Misquitta-Ali, Michael D Wilson, Philip M Kim, Duncan T Odom, Brendan J Frey, and Benjamin J Blencowe. The evolutionary landscape of alternative splicing in vertebrate species. *Science (New York, N.Y.)*, 338(6114):1587–1593, December 2012.

[29] Michael Sammeth, Sylvain Foissac, and Roderic Guigó. A general definition and nomenclature for alternative splicing events. *PLoS computational biology*, 4(8):e1000147, 2008.

[30] Danmin Pan, Kritsanapol Boon-Unge, Piyarat Govitrapong, and Jianhua Zhou. Emetine regulates the alternative splicing of caspase 9 in tumor cells. *Oncology letters*, 2(6):1309–1312, November 2011.

[31] Douglas Hanahan and Robert A Weinberg. Hallmarks of Cancer: The Next Generation. *Cell*, 144(5):646–674, March 2011.

[32] Jason Merkin, Caitlin Russell, Ping Chen, and Christopher B Burge. Evolutionary dynamics of gene and isoform regulation in Mammalian tissues. *Science (New York, N.Y.)*, 338(6114):1593–1599, December 2012.

[33] Liang Chen. Statistical and Computational Methods for High-Throughput Sequencing Data Analysis of Alternative Splicing. *Statistics in biosciences*, 5(1):138–155, May 2013.

[34] Ulrich Braunschweig, Nuno L Barbosa-Morais, Qun Pan, Emil N Nachman, Babak Alipanahi, Thomas Gonatopoulos-Pournatzis, Brendan Frey, Manuel Irimia, and Benjamin J Blencowe. Widespread intron retention in mammals functionally tunes transcriptomes. *Genome research*, 24(11):1774–1786, November 2014.

[35] Yann Christinat, Rafał Pawłowski, and Wilhelm Krek. jSplice: a high-performance method for accurate prediction of alternative splicing events and its application to large-scale renal cancer transcriptome data. *Bioinformatics (Oxford, England)*, 32(14):2111–2119, July 2016.

[36] The GTEx Consortium. The Genotype-Tissue Expression (GTEx) project. *Nature genetics*, 45(6):580–585, 2013.

[37] Miri Danan-Gotthold, Regina Golan-Gerstl, Eli Eisenberg, Keren Meir, Rotem Karni, and Erez Y Levanon. Identification of recurrent regulated alternative splicing events across human solid tumors. *Nucleic Acids Research*, 43(10):5130–5144, May 2015.

[38] Andrew J Gentles, Aaron M Newman, Chih Long Liu, Scott V Bratman, Weiguo Feng, Dongkyoon Kim, Viswam S Nair, Yue Xu, Amanda Khuong, Chuong D Hoang, Maximilian Diehn, Robert B West, Sylvia K Plevritis, and Ash a Alizadeh. The prognostic landscape of genes and infiltrating immune cells across human cancers. *Nature Medicine*, 21(8):938–945, July 2015.

[39] Alexander Koch, Tim De Meyer, Jana Jeschke, and Wim Van Criekinge. MEXPRESS: visualizing expression, DNA methylation and clinical TCGA data. *BMC genomics*, 16(1):636, 2015.

[40] Antonio Colaprico, Tiago C Silva, Catharina Olsen, Luciano Garofano, Claudia Cava, Davide Garolini, Thais S Sabedot, Tathiane M Malta, Stefano M Pagnotta, Isabella Castiglioni, Michele Ceccarelli, Gianluca Bontempi, and Houtan Noushmehr. TCGAbiolinks: an R/Bioconductor package for integrative analysis of TCGA data. *Nucleic Acids Research*, 44(8):e71–e71, May 2016.

[41] Bjarne Johannessen, Anita Sveen, and Rolf I Skotheim. TIN: An R Package for Transcriptome Instability Analysis. *Cancer informatics*, 14:109–112, 2015.

[42] Michael C Ryan, James Cleland, RyangGuk Kim, Wing Chung Wong, and John N Weinstein. SpliceSeq: a resource for analysis and visualization of RNA-Seq data on alternative splicing and its functional impacts. *Bioinformatics (Oxford, England)*, 28(18):2385–2387, September 2012.

[43] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[44] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA, 2015.

[45] Hadley Wickham. testthat: Get started with testing. *The R Journal*, 3:5–10, 2011.

[46] Hadley Wickham and Winston Chang. *devtools: Tools to Make Developing R Packages Easier*, 2016. R package version 1.11.1.

[47] Olaf Mersmann. *microbenchmark: Accurate Timing Functions*, 2015. R package version 1.4-2.1.

[48] Winston Chang and Javier Luraschi. *profvis: Interactive Visualizations for Profiling R Code*, 2016. R package version 0.3.2.

[49] JJ Allaire, Joe Cheng, Yihui Xie, Jonathan McPherson, Winston Chang, Jeff Allen, Hadley Wickham, Aron Atkins, and Rob Hyndman. *rmarkdown: Dynamic Documents for R*, 2016. R package version 0.9.6.

[50] Hadley Wickham, Peter Danenberg, and Manuel Eugster. *roxygen2: In-Source Documentation for R*, 2015. R package version 5.0.1.

[51] M Dowle, A Srinivasan, T Short, S Lianoglou with contributions from R Saporta, and E Antonyan. *data.table: Extension of Data.frame*, 2015. R package version 1.9.6.

[52] Dirk Eddelbuette, Antoine Lucas, Jarek Tuszynski, Henrik Bengtsson, Simon Urbanek, Mario Frasca, Bryan Lewis, Murray Stokely, Hannes Muehleisen, Duncan Murdoch, Jim Hester, Wush Wu, and Thierry Onkelinx. *digest: Create Compact Hash Digests of R Objects*, 2016. R package version 0.6.9.

[53] Yihui Xie. *DT: A Wrapper of the JavaScript Library 'DataTables'*, 2016. R package version 0.2.

[54] Simon Urbanek. *fastmatch: Fast match() function*, 2012. R package version 1.0-4.

[55] Hadley Wickham. *httr: Tools for Working with URLs and HTTP*, 2016. R package version 1.1.0.

[56] Jeroen Ooms. The jsonlite package: A practical and consistent mapping between json data and r objects. *arXiv:1403.2805 [stat.CO]*, 2014.

[57] Arne Henningsen and Ott Toomet. *miscTools: Miscellaneous Tools and Utilities*, 2013. R package version 0.6-16.

[58] Hadley Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1):1–29, 2011.

[59] Hadley Wickham and Romain Francois. *dplyr: A Grammar of Data Manipulation*, 2015. R package version 0.4.3.

[60] Henrik Bengtsson. *R.utils: Various Programming Utilities*, 2016. R package version 2.3.0.

[61] Kun Ren. *rlist: A Toolbox for Non-Tabular Data Manipulation*, 2016. R package version 0.4.6.1.

[62] Dean Attali. *shinyjs: Perform Common JavaScript Operations in Shiny Apps using Plain R Code*, 2016. R package version 0.6.

[63] Douglas H Phanstiel. *Sushi: Tools for visualizing genomics data*, 2015. R package version 1.10.0.

[64] Duncan Temple Lang and the CRAN Team. *XML: Tools for Parsing and Generating XML Within R and S-Plus*, 2016. R package version 3.98-1.4.

[65] Broad Institute of MIT and Harvard. Firebrowse. In review.

[66] Andrew Yates, Kathryn Beal, Stephen Keenan, William McLaren, Miguel Pignatelli, Graham R S Ritchie, Magali Ruffier, Kieron Taylor, Alessandro Vullo, and Paul Flicek. The Ensembl REST API: Ensembl Data for Any Language. *Bioinformatics (Oxford, England)*, 31(1):143–145, January 2015.

[67] Richard J Roberts. PubMed Central: The GenBank of the published literature. *Proceedings of the National Academy of Sciences of the United States of America*, 2001.

[68] Donna Karolchik, Angela S Hinrichs, Terrence S Furey, Krishna M Roskin, Charles W Sugnet, David Haussler, and W James Kent. The UCSC Table Browser data retrieval tool. *Nucleic Acids Research*, 32(Database issue):D493–D496, January 2004.

[69] Markus Ringnér. What is principal component analysis? *Nature biotechnology*, 26(3):303–304, March 2008.

[70] Stephen John Walters. What is a Cox Model?, May 1999.

[71] Spotswood L Spruance, Julia E Reid, Michael Grace, and Matthew Samore. Hazard Ratio in Clinical Trials. *Antimicrobial agents and chemotherapy*, 48(8):2787–2792, August 2004.

[72] Jason T Rich, J Gail Neely, Randal C Paniello, Courtney C J Voelker, Brian Nussenbaum, and Eric W Wang. A practical guide to understanding Kaplan-Meier curves. *Otolaryngology–head and neck surgery : official journal of American Academy of Otolaryngology-Head and Neck Surgery*, 143(3):331–336, September 2010.

[73] Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, New York, 2000.

[74] Terry M Therneau. *A Package for Survival Analysis in S*, 2015. version 2.38.

[75] Shihao Shen, Yuanyuan Wang, Chengyang Wang, Ying Nian Wu, and Yi Xing. SURVIV for survival analysis of mRNA isoform variation. *Nature communications*, 7:11548, 2016.

[76] Todd Neideen and Karen Brasel. Understanding statistical tests. *Journal of surgical education*, 64(2):93–96, March 2007.

[77] Yang Shi, Arul M Chinnaiyan, and Hui Jiang. rSeqNP: a non-parametric approach for detecting differential expression and splicing from RNA-Seq data. *Bioinformatics (Oxford, England)*, 31(13):2222–2224, July 2015.

[78] Francis Sahngun Nahm. Nonparametric statistical tests for the continuous data: the basic concept and the practical use. *Korean journal of anesthesiology*, 69(1):8–14, February 2016.

[79] Brian B Schultz. Levene's Test for Relative Variation. *Systematic Biology*, 34(4):449–456, December 1985.

[80] Alex K Shalek, Rahul Satija, Xian Adiconis, Rona S Gertner, Jellert T Gaublomme, Raktima Raychowdhury, Schraga Schwartz, Nir Yosef, Christine Malboeuf, Diana Lu, John J Trombetta, Dave Gennert, Andreas Gnirke, Alon Goren, Nir Hacohen, Joshua Z Levin, Hongkun Park, and Aviv Regev. Single-cell transcriptomics reveals bimodality in expression and splicing in immune cells. *Nature*, 498(7453):236–240, June 2013.

[81] Boyko Kakaradov, Hui Yuan Xiong, Leo J Lee, Nebojsa Jojic, and Brendan J Frey. Challenges in estimating percent inclusion of alternatively spliced junctions from RNA-seq data. *BMC bioinformatics*, 13 Suppl 6(Suppl 6):S11, 2012.

[82] Olga B Botvinnik, Michael T Lovci, Patrick Liu, Yan Song, and Gene Yeo. Accurate estimation of alternative splicing modalities. `http://yeolab.github.io/flotilla/docs-dev/tutorial/modalities.html`, 2014. Last accessed on 17 July 2016.

[83] Cheng Jia, Yu Hu, Yichuan Liu, and Mingyao Li. Mapping Splicing Quantitative Trait Loci in RNA-Seq. *Cancer informatics*, 14(Suppl 1):45–53, 2015.

[84] Francisco Cribari-Neto and Achim Zeileis. Beta regression in R. *Journal of Statistical Software*, 34(2):1–24, 2010.

[85] Bettina Grün, Ioannis Kosmidis, and Achim Zeileis. Extended beta regression in R: Shaken, stirred, mixed, and partitioned. *Journal of Statistical Software*, 48(11):1–25, 2012.

[86] Juliet Popper Shaffer. Multiple Hypothesis Testing. *Annu. Rev. Psychol.*, 46(1):561–584, January 1995.

[87] Diomidis Spinellis. Version control systems. *IEEE Software*, 22(5):108–109, 2005.

[88] Diomidis Spinellis. Git. *IEEE Software*, 29(3):100–101, 2012.

[89] Josh Paulson. Version Control with Git and SVN. `https://support.rstudio.com/hc/en-us/articles/200532077-Version-Control-with-Git-and-SVN`, 2016. Last accessed on 20 July 2016.

[90] Matteo Carrara, Josephine Lum, Francesca Cordero, Marco Beccuti, Michael Poidinger, Susanna Donatelli, Raffaele Adolfo Calogero, and Francesca Zolezzi. Alternative splicing detection workflow needs a careful combination of sample prep and bioinformatics analysis. *BMC bioinformatics*, 16 Suppl 9(Suppl 9):S2, 2015.

[91] Ann E Loraine, Ivory Clabaugh Blakley, Sridharan Jagadeesan, Jeff Harper, Gad Miller, and Nurit Firon. Analysis and visualization of RNA-Seq expression data using RStudio, Bioconductor, and Integrated Genome Browser. *Methods in molecular biology (Clifton, N.J.)*, 1284(Chapter 24):481–501, 2015.

[92] Wolfgang Huber, Vincent J Carey, Robert Gentleman, Simon Anders, Marc Carlson, Benilton S Carvalho, Hector Corrada Bravo, Sean Davis, Laurent Gatto, Thomas Girke, Raphael Gottardo, Florian Hahne, Kasper D Hansen, Rafael A Irizarry, Michael Lawrence, Michael I Love, James MacDonald, Valerie Obenchain, Andrzej K Oleś, Hervé Pagès, Alejandro Reyes, Paul Shannon, Gordon K Smyth, Dan Tenenbaum, Levi Waldron, and Martin Morgan. Orchestrating high-throughput genomic analysis with Bioconductor. *Nature methods*, 12(2):115–121, February 2015.

[93] Dirk Eddelbuettel and Romain Fran. Rcpp: Seamless R and C ++ Integration. *Journal Of Statistical Software*, 40(8):1–18, 2011.

[94] Simon Anders, Davis J McCarthy, Yunshun Chen, Michal Okoniewski, Gordon K Smyth, Wolfgang Huber, and Mark D Robinson. Count-based differential expression analysis of RNA sequencing data using R and Bioconductor. *Nature protocols*, 8(9):1765–1786, September 2013.

[95] Laurent Gatto and Andy Christoforou. Using R and Bioconductor for proteomics data analysis. *Biochimica et biophysica acta*, 1844(1 Pt A):42–51, January 2014.

[96] Hadley Wickham. *Advanced R*. CRC Press, September 2015.

[97] John M Chambers. Object-Oriented Programming, Functional Programming and R. *Statistical Science*, 29(2):167–180, 2014.

[98] Hadley Wickham. *R Packages*. "O'Reilly Media, Inc.", March 2015.

[99] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, September 2012.

[100] John Hughes. Why Functional Programming Matters. *The Computer Journal*, 32(2):98–107, February 1989.

[101] Ramnath Vaidyanathan, Yihui Xie, J J Allaire, Joe Cheng, and Kenton Russell. *htmlwidgets: HTML Widgets for R*, 2015.

[102] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009.

[103] Fiona Cunningham, M Ridwan Amode, Daniel Barrell, Kathryn Beal, Konstantinos Billis, Simon Brent, Denise Carvalho-Silva, Peter Clapham, Guy Coates, Stephen Fitzgerald, Laurent Gil, Carlos García Girón, Leo Gordon, Thibaut Hourlier, Sarah E Hunt, Sophie H Janacek, Nathan Johnson, Thomas Juettemann, Andreas K Kähäri, Stephen Keenan, Fergal J Martin, Thomas Maurel, William McLaren, Daniel N Murphy, Rishi Nag, Bert Overduin, Anne Parker, Mateus Patricio, Emily Perry, Miguel Pignatelli, Harpreet Singh Riat, Daniel Sheppard, Kieron Taylor, Anja Thormann, Alessandro Vullo, Steven P Wilder, Amonida Zadissa, Bronwen L Aken, Ewan Birney, Jennifer Harrow, Rhoda Kinsella, Matthieu Muffato, Magali Ruffier, Stephen M J Searle, Giulietta Spudich, Stephen J Trevanion, Andy Yates, Daniel R Zerbino, and Paul Flicek. Ensembl 2015. *Nucleic acids research*, 43(Database issue):D662–9, January 2015.

[104] Cathy H Wu, Rolf Apweiler, Amos Bairoch, Darren a Natale, Winona C Barker, Brigitte Boeckmann, Serenella Ferro, Elisabeth Gasteiger, Hongzhan Huang, Rodrigo Lopez, Michele Magrane, Maria J Martin, Raja Mazumder, Claire O'Donovan, Nicole Redaschi, and Baris Suzek. The Universal Protein Resource (UniProt): an expanding universe of protein information. *Nucleic Acids Research*, 34(90001):D187–D191, 2006.

[105] Donna Karolchik, Galt P Barber, Jonathan Casper, Hiram Clawson, Melissa S Cline, Mark Diekhans, Timothy R Dreszer, Pauline a Fujita, Luvina Guruvadoo, Maximilian Haeussler, Rachel a Harte, Steve Heitner, Angie S Hinrichs, Katrina Learned, Brian T Lee, Chin H Li, Brian J Raney, Brooke Rhead, Kate R Rosenbloom, Cricket a Sloan, Matthew L Speir, Ann S Zweig,

David Haussler, Robert M Kuhn, and W James Kent. The UCSC Genome Browser database: 2014 update. *Nucleic Acids Research*, 42(Database issue):D764–70, January 2014.

[106] Thomas A Down, Matias Piipari, and Tim J P Hubbard. Dalliance: interactive genome viewing on the web. *Bioinformatics (Oxford, England)*, 27(6):889–890, March 2011.

[107] Jian Shen, Weizhi Wang, Jindao Wu, Bing Feng, Wen Chen, Meng Wang, Jincao Tang, Fuqiang Wang, Feng Cheng, Liyong Pu, Qiyun Tang, Xuehao Wang, and Xiangcheng Li. Comparative proteomic profiling of human bile reveals SSP411 as a novel biomarker of cholangiocarcinoma. *PloS one*, 7(10):e47476, 2012.

[108] Mathias Meyer. Continuous Integration and Its Tools. *IEEE Software*, 31(3):14–16, 2014.

[109] Carl J Mueller, Dan Tamir, Oleg V Komogortsev, and Liam Feldman. An Economical Approach to Usability Testing. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, pages 124–129. IEEE, 2009.

[110] Ajay Bandi and Phil Heeler. Usability testing: A software engineering perspective. In *Human Computer Interactions (ICHCI), 2013 International Conference on*, pages 1–8. IEEE, 2013.

[111] Stephanie Rosenbaum and Laurie Kantner. Field Usability Testing: Method, Not Compromise. In *2007 IEEE International Professional Communication Conference*, pages 1–7. IEEE, 2007.