Software Design Document

**Victoria State Accident Data Analysis**

**S5254936 – Bebin Roy**

**S5182075 – Atticus Burgess**

**S5299401 – Ansh**

Table of Contents

[1.0 System Vision 3](#_Toc144379971)

[1.1 Problem Background 3](#_Toc144379972)

[1.2 System Overview 3](#_Toc144379973)

[1.3 Potential Benefits 3](#_Toc144379974)

[2.0 Requirements 4](#_Toc144379975)

[2.1 User Requirements 4](#_Toc144379976)

[2.2 Software Requirements 4](#_Toc144379977)

[2.3 Use Cases & Use Case Diagrams 5](#_Toc144379978)

[3.0 Software Design and System Components 6](#_Toc144379979)

[3.1 Software Design 6](#_Toc144379980)

[3.2 System Components 6](#_Toc144379981)

[3.2.1 Functions 6](#_Toc144379982)

[3.2.2 Data Structures / Data Sources 8](#_Toc144379983)

[3.3.3 Detailed Design 9](#_Toc144379984)

[4.0 User Interface Design 11](#_Toc144379985)

[4.1 Structural Design 11](#_Toc144379986)

[4.2 Visual Design 12](#_Toc144379987)

# 1.0 System Vision

## Problem Background

Road accidents have been a persistent issue in Victoria, Australia, with numerous fatalities and injuries reported over the years. The State of Victoria has accumulated data from 2015 to 2020, detailing various aspects of these accidents. However, the raw data, while extensive, doesn't provide immediate insights or patterns that can be used for preventive measures or policymaking. There's a need for a tool that can analyse, visualise, and provide insights from this data to various stakeholders, including accident commissions, insurance companies, and the public.

## System Overview

The proposed system will be a data analysis and visualization tool tailored for the road crash statistics dataset of Victoria. This tool will offer a user-friendly graphical interface, allowing users to filter and understand specific aspects of the data, such as crash types, locations, conditions, and more. The software will not only address the predefined analysis tasks but will also introduce an additional unique insight feature, derived from the dataset, to provide a comprehensive understanding of the road safety situation in Victoria. 

## Potential Benefits

**Informed Decision Making:** Policymakers can use the insights from the tool to make informed decisions regarding road safety measures, infrastructure development, and public awareness campaigns.

**Insurance Insights:** Insurance companies can gain a better understanding of high-risk areas, types of accidents, and other relevant data points to adjust their policies and premiums accordingly.

**Public Awareness:** The general public can access and understand the data, leading to increased awareness about high-risk areas, times, and other factors. This can potentially lead to safer driving habits.

**Resource Allocation:** By identifying high-risk areas and times, resources such as ambulances, police, and emergency services can be allocated more efficiently.

**Continuous Improvement:** As the tool will be continuously updated with new data, it will evolve and provide more accurate and timely insights, leading to a dynamic approach to road safety in Victoria.

# 2.0 Requirements

## 2.1 User Requirements

The software will be used as public domain and the development team will accept donations as payment. The software is being made for the purpose of the public to gain an extensive insight on dangers and reasons for vehicular accidents in the state of Victoria. With the user demographic of regular citizens some maybe not being as tech savvy the software will be as easy to use as possible.

Presented below are the requirements that must be present for the user to gain value from the software:

User Requirement 1.1 – the data will initially be sorted by date (oldest incident appearing first), allowing for users to become familiarised with the data structure and values.

User Requirement 1.2 – the ability to apply various filters to the dataset will be most crucial to create value for the user. The filters will be able to sort incidents by:

* Specified dates
* Time of day
* Main cause of incident,
* If alcohol was involved
* Number of casualties

User Requirement 1.3 – the ability to produce graphs with previously applied filters to compare data. These graphs will allow the user to gain a much better understanding of the relationships between values in the desired time period.

User Requirement 1.4 – as the program will be public domain and be available for anyone to use it must be simple and easy to understand. All headings, fonts and colour choices in the user interface must be of adequate size and allow the user to have a clear understanding of their choices and goals.

## 2.2 Software Requirements

The software requirements are linked with specific tasks the software must be capable of producing. There must not be any bugs or errors in the final deliverables for the client.

Software Requirement 1.1 – The program shall be easy and simple to navigate with clear headings and buttons. The software will abide by suitable visual design standards.

Software Requirement 1.2 – The default way the data shall be sorted by is by date.

Software Requirement 1.3 – The program shall accept only single words or “strings” that will henceforth interact with the dataset.

Software Requirement 1.4 – The program shall accept numerical digits (floats and strings) when the user is prompted that will henceforth interact with the dataset.

Software Requirement 1.5 – Once provided with desired requests the program shall automatically produce specified graphs and tables.

Software Requirement 1.6 – The user shall be able to export the desired graphs or tables as a simple file name and a destination on their local machine.

## 2.3 Use Cases & Use Case Diagrams

**Use Case 1** – the user opens the application and wants to view data from March 2016 – August 2016. The dataset that will be by default presented to them will be sorted by oldest to newest. The user will select the first interactable dropdown menu labelled “Date From” then the second dropdown menu labelled “Date To” to specify their desired time period. After entering their desired dates, the user will click the large blue “apply” button which will henceforth display all incidents within that timeframe.

**Use Case 2 –** the user as detailed above would like to furthermore investigate the relationships in the selected time frame. the user would like to apply the filter if there was alcohol present in the accident. The user applies the filter in the second menu tab labelled “Alcohol influence” and ticks the “Yes” label. The data is now properly filtered and displayed.

**Use Case 3 –** the user would like to apply the hit and run filter to all data entries. They are greeted by the default program splash screen and click on the “key word description” drop down menu and applies the hit and run filter. This will display every data entry with a hit and run keyword and output the entries to the display field.

**Use Case 4 –** the user wants to determine how many accidents had 2 casualties. They apply “2” into the dropdown menu titled “Casualty amount” and all data entries are displayed.

A diagram of a accident analysis program

Description automatically generated**Use Case 5 –** the user would like to search for all accidents in 2018 that have been influenced by alcohol and had 1 casualty. Upon opening the program, they will select the first dropdown menu labelled “Date From” and select January 1st then select the “Date To” and select December 31st. Next the user would apply the “Alcohol influence” check and finally in the next dropdown menu labelled “Casualty amount” select the “1” option. The desired data sets will be displayed in the display area.

# 3.0 Software Design and System Components

## 3.1 Software Design

A diagram of a computer process

Description automatically generated

The application functions as a basic database-style utility. It commences by presenting the initial dataset and has the capability to extend further. It can incorporate filters and search phrases, or even specifications for graphical representation, subsequently modifying how the data is exhibited. The filter options can subsequently be reset, or the swift display of data can be activated and deactivated at will. The transitions between these various states are fluid, enabling seamless movement between them as required. the development group will use PyCharm to code all portions of the product. The group is familiar with the PyCharm environment and its features.

## 3.2 System Components

### 3.2.1 Functions

Here is an initial compilation of software functions. Each function's details encompass a concise overview, a roster of input parameters, a listing of resultant influences, and an elucidation of the function's output.

The subsequent functions are covered:

**loadData:**

• The load Data function is triggered during program initiation, orchestrating the integration of the primary data array into the program, and priming it for presentation.

• This function necessitates no user inputs and triggers automatically upon program launch.

• It furnishes the data sourced from the database as its output.

**displayData:**

• The Display Data function is executed whenever data is loaded or filters undergo modifications, facilitating the exhibition of presently chosen datasets.

• This function operates automatically without the need for user inputs.

• It showcases the presently filtered data on the program's screen.

**filterData:**

• The Filter Data function is responsible for modifying the displayed data to harmonize with an array of chosen filters provided by the user, permitting data filtration based on factors like date or location.

• It necessitates keywords in the form of strings, which effectuate the transformation of variables to activate pre-established filters.

• Filters that undergo modifications will amend the data returned for display through the manipulation of variables.

• This function yields a Boolean value of 'true' if the filter is currently active.

**searchData:**

• The Search Data function is invoked when a user initiates a data search using particular keywords. It will systematically scan through the dataset and identify instances of the specified keywords.

• The function relies on keywords, expressed as strings, which guide its search parameters. • The function creates duplicates of dataset objects that encompass the identified keywords and assembles them into a fresh array for subsequent presentation.

• Its outcome is a novel array of datasets earmarked for display.

**graphData:**

• The Graph Data function undertakes the conversion of chosen data into graphical representations designed to furnish elucidation of the dataset.

• The preferred graph format is fed into the function as a sequence of keywords.

• The function reconfigures the way the data is exhibited to the user.

• Its yield consists of methodically structured and graphically depicted data derived from the main data array, contingent on user-provided input.

**clearFilters:**

• The Clear Filters function is designed to eradicate all currently active filters.

• It requires input in the shape of a command, which is a string.

• The function's operation entails the complete elimination of all filters influencing the data, thereby restoring the data to its unaltered presentation sourced from the primary data array.

• Upon execution of the specified command, the function yields a Boolean value of 'true'.

**hitandrunToggle:**

• The hitandrunToggle function operates by activating or deactivating the incorporation of data associated with hit-and-run incidents.

• It accepts a Boolean input that determines whether the hit-and-run filter is in an activated or deactivated state.

• This function induces modifications in the displayed data. When the filter is off, all data, irrespective of hit-and-run status, is showcased. Conversely, when the filter is toggled on, data linked to hit-and-run accidents is concealed.

• The outcome of activating the toggle returns a Boolean value of 'true'.

### 3.2.2 Data Structures / Data Sources

**Data:**

• The Data component consists of an array.

• This array serves as the primary compilation of all accident-related data. It holds the status of being the master array from which the displayed data is extracted after undergoing filtration.

• Data members within the array encompass all datasets pertaining to accidents.

• The search, filter, graph, and hit-and-run toggle functions collectively draw data from this array for the purpose of relaying it to the display data array.

**Dataset:**

• The Dataset is conceptualized as an object.

• It embodies distinct sets of accident-related data, each existing within the overarching data array. These individual datasets are duplicated into the display array in alignment with the preferences articulated by the user.

• The Dataset object is characterized by its constituents, representing the attributes and properties associated with accidents.

• While no dedicated functions explicitly leverage this dataset, it serves as the quintessential data encapsulated within both the display and overall data arrays. Its role is to document the particulars of accidents.

**Display Data:**

• Display Data is conceived as an array.

• It houses the existing data that is being showcased within the program.

• Membership in this array is reserved for datasets that have successfully navigated through the processes of search and filtration.

• While no specific functions are tailored to this array, its essence lies in featuring the data post the application of various filters and functions. It represents the ultimate data display after all other transformations have taken place.

**Filters:**

• Filters are conceptualized as an array.

• They exert influence over the data display process by dictating which filters are to be applied to the data.

• The constituents of this array comprise the filter variables that function as determinants for the presence or absence of diverse filters upon the data.

• The filter function relies on the data contained within this array to execute its tasks.

### 3.3.3 Detailed Design

Pseudocode is a major aspect of all coding languages and crucial in communicating ideas with team members in an “unformal” way. When developing our product our group will abide by strict guidelines when writing pseudocode. These guidelines are:

* Pseudocode is placed above the code line it is in relation to.
* Multiline/Single line comments must be used where applicable.
* Pseudocode must be descriptive and well written, allowing team members to understand clearly what is being conveyed.

**loadData:**

Must retrieve data from desired online database. Must store data in a data array as an object.

**displayData:**

Pulls the data array and adds each object to be displayed in the designated display area. The data by default will be displayed from oldest to newest.

**filterData:**

Iterates over all data for entries of a specific key word e.g., “crash”, “side swipe”, “pedestrian”. Must clear all display data and start fresh when wanting to filter for keywords. Once matches are found it will display in the display area.

**searchData:**

Starts a new empty array and searches through data for desired keywords and displays the array into the display area.

**graphData:**

Once desired information has been filtered and displayed in target area, data is formatted and displayed as a column or line graph.

**clearFilters:**

Empties all arrays with filters applied and redisplays the default master data set as if program was initially running.

**hitandrunToggle:**

Applies the hit and run filter to the master array and displays the entries in the display area.

# 4.0 User Interface Design

In the initial stages of our interface design, we employed a combination of modern design tools and methodologies to ensure a user-centric approach. Our primary objective was to create an interface that is both intuitive and efficient, catering to the diverse needs of our users.

Tools used:  
Draw.io: This digital design toolkit allowed us to create a blueprint of our interface, laying out the basic structure and elements.   
  
Figma: Leveraging Figma's collaborative features, our team was able to work simultaneously on design prototypes, ensuring real-time feedback and iterations. Its vector-based tools also facilitated the creation of high-fidelity mock-ups.   
  
Adobe XD: For interactive prototyping, we turned to Adobe XD. This enabled us to simulate the user experience, testing the flow and transitions between different sections of our application.

## 4.1 Structural Design

The structure of this product is designed to provide users with a seamless experience, allowing them to easily navigate through the data, apply necessary filters, visualize the data in tables or graphs, and export the results. The choices made in this design prioritize user convenience and efficiency, ensuring that users can quickly access and analyse the data they need.

## 4.2 Visual Design

A screenshot of a computer

Description automatically generated

**Visual Design Justification:**

Our visual design prioritizes clarity, efficiency, and user-friendliness. The layout was crafted to ensure that users can quickly access and interact with the most crucial features.

* **Date and Time Selector at the Top**: Placing the date and time selector at the top is to ensure the users are able to filter to apply when analysing data. Its prominent position ensures that users can quickly set their desired time frame simply.
* **Alcohol Influence Filter**: Given the significance of understanding the impact of alcohol in accidents, a straightforward 'yes' or 'no' toggle was integrated. This choice simplifies the filtering process, allowing users to instantly view accidents influenced by alcohol.
* **Casualty Dropdown and Keyword Description**: These features are essential for users who want a more filtered analysis. By providing a dropdown for the number of casualties and a keyword description, users can tailor their data view to specific scenarios or types of accidents. This may include locations and maybe vehicle descriptions.
* **Apply and Reset Buttons**: These buttons are placed near the filters for immediate action. The 'apply' button ensures users can confirm their choices, while the 'reset' button offers a quick way to start afresh, enhancing user experience by offering flexibility.
* **Graph and Table View on the Right**: Visual data representation is vital for intuitive understanding. By placing the graph and table view on the right, users can immediately see the results of their filters, making data analysis more seamless.
* **Export Data Button**: Recognizing the need for users to utilize the data outside the platform, an export button is conveniently placed. This ensures that users can easily take their insights and share or further analyse them in different environments.

In conclusion, every element of our wireframe is designed with the user's needs in mind. The layout and visual elements are not just aesthetic choices but are rooted in the objective of making data analysis as intuitive and efficient as possible for our users.