TECHNISCHE UNIVERSITÄT
CHEMNITZ

# Efficient Generation of Standard Customer Reports for Airbag Simulation Results

**Master Thesis**

Submitted in Fulfillment of the
Requirements for the Academic Degree
M.Sc.

Dept. of Computer Science
Chair of Computer Engineering

Submitted by: Sagar Jayanthi
Student ID: 673217
Date: 24.04.2023

University supervisors: Prof. Dr. Dr. h. c. Wolfram Hardt, Dr. Ariane Heller
Company supervisor: Marcus Lehmann

# Abstract

Passive safety systems like airbags have significantly improved road safety. These occupant safety systems help in reducing the severity of injuries, and save lives in the event of a road accident. The airbag systems must be configured correctly to minimize the impact of collision and protect the occupants. To configure the airbag, test crashes are performed and data is recorded. This data is simulated to find out appropriate parameters for the airbag deployment. The airbag simulation results are stored into databases. Airbag application tools are used to handle the data stored in databases. The airbag simulation results must be extracted efficiently and required computations needs to be performed. This data is then stored to reports. RSDBnext is an airbag application tool, it stands for Result Database next generation. This tool is used for extraction of data from the database. The RSDBnext tool should be adapted to generate Standard Customer Reports. These reports are to be generated based on customer requirements. The existing methodology to generate Standard Customer Reports used Excel macros, which took a lot of time to generate the reports. This method was complex and unstable. Hence, a new methodology was proposed without using macros. In the proposed method, an XML file and XSLT StyleSheet were used to generate the report in Excel using C# with Visual Studio. This approach reduces report generation time, and overcomes the drawbacks of the previous approach. From the results, this methodology to generate reports is faster, easier, and more reliable.

**Keywords: Airbag, Crash, Simulation, Standard Customer Reports**

# Acknowledgement

To be filled

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | | | |
|---|---|---|---|
| **NHTSA** | National Highway Traffic Safety Administration | **XML** | Extensible Markup Language |
| **AB** | Airbag | **HTML** | Hyper Text Markup Language |
| **ACU** | Airbag Control Unit | **CDR** | Crash Data Retrieval |
| **ECU** | Electronic Control Unit | **XSLT** | Extensible Stylesheet Language Transformation |
| **RTTF** | Required Time To Fire | **JSON** | JavaScript Object Notation |
| **SUV** | Sports Utility Vehicle | **VBA** | Visual Basic for Applications |
| **ASZ** | Automatic Suppression Zone | **ARXML** | AUTOSAR Extensible Markup Language |
| **CI** | Continuous Integration | **RSDB** | Result Database |
| **RTC** | Rational Team Concert | **RSDBnext** | Result Database Next Generation |
| **AUTOSAR** | Automotive Open System Architecture | **IDE** | Integrated Development Environment |
| **OEM** | Original Equipment Manufacturer | **API** | Application Programming Interface |
| **RTE** | RunTime Environment | **TMin** | Minimum Fire Time |
| **BSW** | Basic Software | **TMax** | Maximum Fire Time |
| **CAN** | Contoller Area Network | **RSDBnext** | Result Database Next Generation |
| **ADAS** | Advanced Driver Assistance Systems | **NF** | No Fire |
| **LED** | Light Emitting Diode | **FF** | Full Fire |
| **CPU** | Central Processing Unit | **AI** | Artificial Intelligence |
| **GUI** | Graphical User Interface | | |

# 1 Introduction

One of the most serious public safety problems is road accidents. The number of deaths caused by automobile accidents are increasing rapidly and measures need to be taken to reduce it. In the recent years, the road traffic has been increasing. The number of road accidents is increasing along with the rise in traffic. These incidents can occur for a variety of causes, including vehicle malfunction, poor road conditions, driver errors, and other external variables such as weather. These accidents can range from minor injury to serious injury or fatal death even. World Health Organization states that there are approximately 1.35 million traffic deaths every year, and many more millions of non-fatal injuries.

According to the European Commission, over 22,800 individuals were killed in the European Union in 2019. Additionally 120,000 people were seriously injured in road accidents. Although, deaths have been reduced over the years, road accidents still remain a major concern. Moreover, the European Union has also implemented several regulations to improve the road safety, such as installation of advanced safety features like emergency braking systems and speed assistance systems. [1]

Reducing these road accidents is a complex issue and it must be handled by improving road safety. Some ways that can improve road safety are as following:

- **Improve the Road Infrastructure:** By improving the road conditions and infrastructure, such as wider roads, improved lighting, and better road signs can help reduce the number of accidents.

- **Promoting Safe Driving Practices:** Educating the driver about safe driving practices, such as to avoid distractions while driving, following the traffic laws and driving safely improves safety on the road. These practices can reduce the risk of road accidents.

- **Improved Emergency Response:** Emergency response systems must be effective. In the case of an automobile accident, an emergency response system can assist lessen the severity of an occupant's injuries. This can also reduce the risk of fatalities.

- **Enhancing the Vehicle Safety Features:** The usage of safer vehicles must be encouraged. For e.g., vehicles which have advanced safety features like automatic emergency braking system and advanced passive safety system like airbags. Employing these safety systems can mitigate the number of crashes. Passive safety systems reduce the severity of injuries during a crash.

Preventing these road accidents can be done by educating the driver as well as imposing stricter traffic rules and regulations. Overall, road safety measures such as improving road infrastructure are employed to reduce accidents. The safety systems of the vehicle needs to be improved constantly to protect the road users.

## 1.1 Background

When a collision occurs, passive safety features can protect the car's occupants and lessen the severity of injuries. In the following section, the airbag systems (airbags and airbag control unit) are discussed. After the airbag systems section, the airbag simulations and the way their results are obtained is discussed.

### *1.1.1* Airbags and Airbag Control Unit

Airbags are a passive safety feature meant to protect automobile occupants in the event of an accident. The airbags are composed of strong and flexible material and are designed to inflate quickly. Airbags are designed to cushion the occupants of the car. When a collision occurs, then the sensors detect the impact of the collision. Along with the impact of the collision, the deceleration of a car is also detected. The airbags are then deployed. This process of airbag deployment, where in the airbag gets inflated starts and takes place within a few milliseconds.

Airbags have significantly reduced the number of fatalities and also the severity of injuries in automobile accidents. This airbags are designed to be used with another passive safety system, the seatbelts. The airbags are installed at various places in a vehicle. The position of airbag installation depends on the manufacturer and the model of the automobile. The airbag inflates and the seatbelts are used to reduce the impact of the crash on the occupants in the vehicle. The most important and common location of an airbag is in the vehicle’s steering wheel. The airbag is folded inside the steering wheel as shown in Figure 1.1. In addition to the steering wheel, airbags are usually located in the dashboard and seat backrests.

**Figure 1.1: Folded Airbag in the Steering Wheel [1]**

Frontal airbags have saved more than 50,000 lives according to the National Highway Traffic Safety Administration (NHTSA). The frontal airbags are intended to deploy in the event of a frontal car crash. Figure 1.2 shows the frontal airbag which is inflated during a car crash. The activated belt tensioner which is triggered during a crash can also be seen. The belt tensioner is used to prevent the occupant from colliding with the dashboard. Hence, the occupant is safe from head injuries or deaths during a car accident.

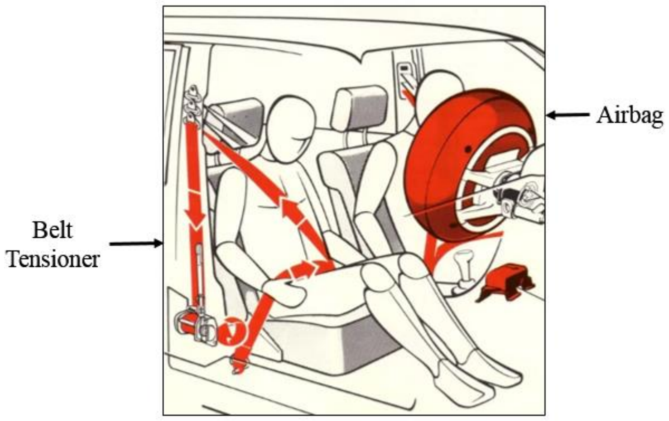


**Figure 1.2: Trigger of Airbag and Belt Tensioner [2]**

Different airbags are designed to deploy during various crash scenarios. These scenarios are mainly based on the location of the impact. For example, if a car collides with the front of another car, then the type of crash is classified into a frontal crash. There are also other types of crashes like rollover crash and side impact crash. Some of the possible crash scenarios are listed below:

- **Frontal Crash:** This type of collision occurs when a car collides with another vehicle or any stationary item. To protect the driver's life in case of this type of crash, the front airbags are placed in the steering wheel. The location of the front passenger airbag is the dashboard of the car. It is intended to safeguard the front passenger. An example of frontal collision can be seen in Figure 1.3.

**Figure 1.3: Frontal Car Crash [3]**

- **Rollover Crash:** When a car rolls over or topples, then this type of crash is classified as rollover crash. For example, when a car falls down a cliff and rolls over or when the car topples over any object on the road. To protect the occupant from injuries in a rollover collision, curtain airbags are deployed. An example of a rollover crash can be seen in Figure 1.4.

**Figure 1.4: Rollover Crash [3]**

- **Side Impact Crash**: Side impact crashes occur when a vehicle collides with another vehicle or any object on the sides of a car. These type of collisions are more dangerous. This is because there is less protection on the sides of a car compared to the front or back side. These crashes may occur on the highways or at a road intersection. Curtain airbags and side airbags are designed to protect occupants during this type of collision.

- **Rear-end Collision:** When a vehicle collides with the back side of another vehicle, then this crash is classified as a rear-end collision. Some vehicles are fitted with rear end airbags to protect the passengers during this type of crash. The speed of the vehicle is important during this kind of crash. For e.g., when the car is at high speeds, the occupant may suffer head and spinal cord injuries.

- **Offset Crash:** When the front of a vehicle collides with another vehicle at any other angle rather than a head-on collision, then it is an offset crash. Based on the conditions of an accident, the decision is made whether the frontal and side airbags must be activated or not.

These are the most typical types of car crashes which can occur in a road accident. Airbags are designed and placed at various locations of the vehicle. Most commonly used airbags are front driver airbag, front passenger airbag and curtain airbags as shown in Figure 1.5. Depending on the car's model, a vehicle may have multiple airbags installed. Typically, the number of airbags installed in a car is six to ten.

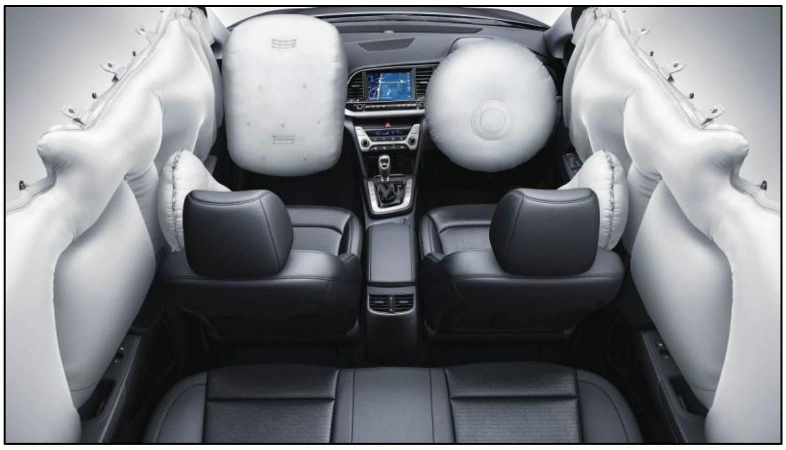


**Figure 1.5: Front, Side and Curtain ABs in a Car [4]**

The deployment of an airbag is carried out in 5 stages, as shown in Figure 1.6. Each stage of airbag deployment happens within a few milliseconds. The process of airbag deployment starts when a crash is detected by the sensor, and then the following stages are activated. The 5 stages of airbag deployment are as follows:

- **Stage 1:** The first step is the crash detection stage where the sensors of a vehicle detect a crash. This stage happens within 3 milliseconds. The accelerometer, impact sensor and other sensors involved detect a crash.

- **Stage 2:** The second stage is the evaluation stage. During this stage, the crash signal captured by the sensors is processed. The processing of the crash signal takes place within 15 milliseconds. The severity of the collision is determined. A decision is made on which passive safety system needs to be activated. Based on this decision, the airbag inflators and other safety systems receive a signal whether the airbag should be deployed or not.

- **Stage 3:** Airbags are deployed during this stage. If the decision is to fire the airbags, then the inflator causes a chemical reaction that emits nitrogen gas or argon gas. This gas is released in a controlled manner to inflate the airbag. During this stage, the inflation of airbags is started.

- **Stage 4:** This is the expansion stage, where the airbag reaches to its full size and shape in approximately 20-30 milliseconds. The airbag is designed to cushion the occupant in order to reduce the risk of injury in case of a crash. The passenger motion energy is fully absorbed by the airbags during this stage.

- **Stage 5:** Deflation stage of the inflated airbag happens in this stage. After the passenger motion is fully absorbed, the next stage is activated. The deflation of airbags is started, where the gas filled inside the airbag deflates. The deflation of the gas occurs through the provided vents. This happens almost immediately within 20 milliseconds. The deflation of airbags allows the occupant to exit the vehicle without any difficulties. Airbags are deflated to also ensure that the additional injuries caused by airbags are reduced.

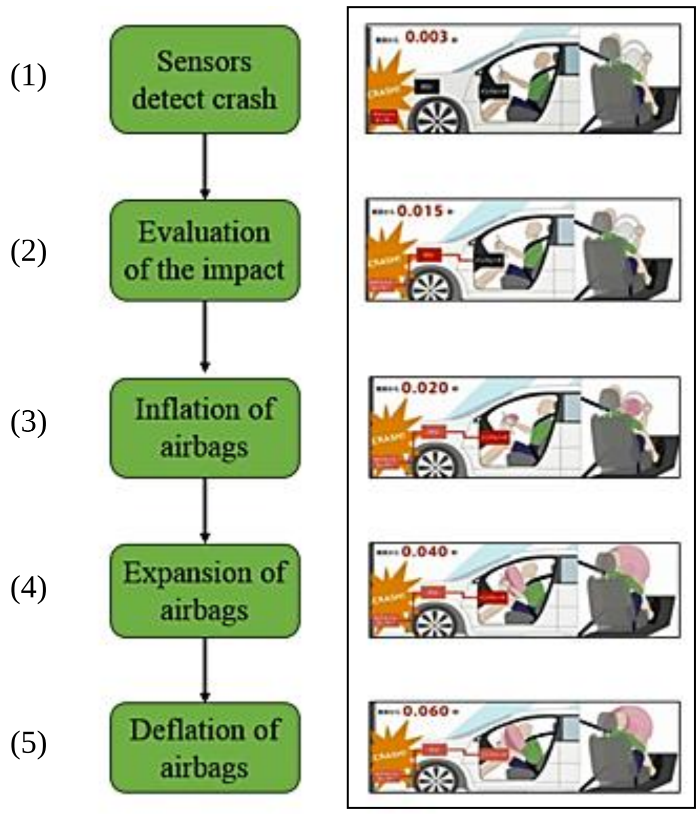


**Figure 1.6: Airbag Deployment Stages [5]**

The deployment of airbags, also known as the fire of airbags is controlled by various conditions. These may be speed of the car, angle of impact and the type of crash. After considering all these conditions, the Airbag Control Unit decides the airbag firing time appropriately. The fire time of the airbags is very important. There are several restrictions if the airbags are not deployed at the correct time. The negative effects of airbag firing at wrong times is shown in Table 1.1.

| Deployment of Airbags | Negative Effects on Occupants |
|---|---|
| AB does not deploy or deploys too early | • Causes major injuries during an accident |
| AB deploys accidentally | • Cause injuries<br>• For e.g., losing control of the car, leading to another accident |
| AB deploys late (after a crash) | • Leads to even greater injuries compared to the case where airbags are not deployed at all<br>• For e.g., the occupant may not be able to escape from the vehicle immediately after an accident |

**Table 1.1: Negative Effects of AB Deployment at the Wrong Time**

To avoid the negative effects of wrong airbag deployment, the airbag should be deployed at an optimal time. This deployment of airbags is controlled by a control unit, which is known as an Airbag Control Unit (ACU). Each car has an ACU, which is located near the dashboard of the car or at the center of the vehicle. It is connected to different sensors throughout the vehicle. These sensors detect the severity and the kind of impact. Figure 1.7 shows the first developed Airbag Control Unit. It was patented by Bosch in the year 1976. [2]

The ACU measures the deceleration occurring when a crash takes place. So, the airbags can be fired at the correct time in case of a crash. ACU can also be used to trigger automatic belt tensioners which ensure that the occupants are held in the same position during a collision. Hence, the occupants do not collide into the dashboard or steering wheel and are protected from severe injuries.

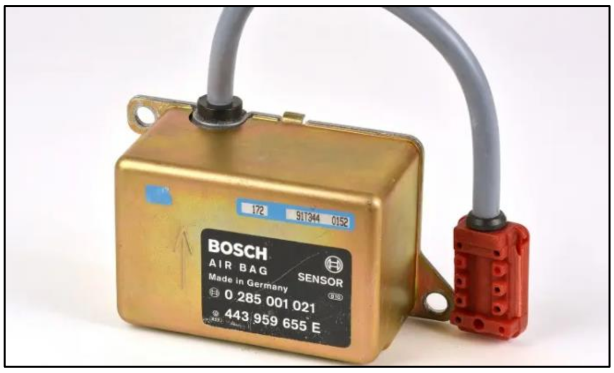


**Figure 1.7: First Generation Airbag Control Unit [2]**

When a crash occurs, the sensors of a car transmit a signal to the airbag control unit which processes the signal. After evaluating the received crash signal, it then decides if the airbags should be deployed or not. In the case of deployment of an airbag, a signal is sent to the airbag inflator. It is a device that is responsible for the inflation of airbags. It is a device located in the dashboard, and is filled with a gas like nitrogen or argon. This gas is then used to inflate the airbags. When the airbag inflator receives a signal from the ACU, a chemical reaction is caused. This inflates the airbag in just a couple of milliseconds. These airbags are made of a flexible, strong material. They are designed to inflate as quickly as possible and act as a cushion to the occupants.

The ACU in the vehicle determines when the airbags should be deployed. ACU also ensures that airbags are deployed in a timely manner, and in a regulated way. Airbag deployment is determined by the ACU based on a number of criteria. These criterion may include the severity of impact, kind of collision, the location and movement of occupants.

In addition to controlling the deployment of airbag systems, the ACU performs various other functions as a part of the passive safety system. Responsibilities of the ACU in a vehicle are as follows:

- **Monitoring the Airbag System:** The ACU monitors the status of the airbag system constantly, to ensure its proper functionality. If there is a problem with the airbag system, then the driver is alerted by turning on a warning light on the dashboard.

- **Interfacing with other Systems:** ACU is responsible for communicating with the electronic control unit, also known as ECU and the brakes to coordinate the response to a collision.

- **Storing Diagnostic Data:** The ACU records information about any past airbag deployment or any problems with the airbag systems detected. This information can be accessed using a diagnostic tool to fix any problems with the airbag system.

- **Controlling the Deployment of other Safety Systems:** Safety systems like belt pre-tensioners, stability control systems are also deployed in the event of a collision. The deployment of these safety systems are monitored by the ACU.

These are the responsibilities of an ACU in an automotive. The airbags and airbag control units are constantly being improved with the rising necessity. Together with the seatbelts and belt tensioners, airbags are one of the most effective passive safety system of an automotive.

When a crash occurs, the crash output is detected and generated by the ACU. This device sends out crash signals to other devices like ECU. The typical applications for crash output are as follows:

- ➢ Activate brakes
- ➢ Deactivate fuel pump
- ➢ Deactivate high voltage batteries
- ➢ Open door locks
- ➢ Turn on hazard flasher
- ➢ Send out an emergency call

Along with the firing of the passive safety systems, the ACU is also responsible for performing the crash output.

Overall, the ACU plays a crucial role in controlling the complete airbag system, as it is responsible for the deployment of airbags and other safety systems. It ensures that airbags are deployed effectively in case of a collision.

### *1.1.2* Airbag Simulation Results

The airbags must fire at the right time to save the lives of the occupants and minimize the injuries caused. The firing of an airbag must be configured correctly. These fire times vary on the type of crash scenario. The ACU of the vehicle sends a signal at the firing time to the particular safety device. To identify the correct fire times of these passive safety systems, various simulations are performed. During these simulations, the fire times for each case is determined.

Bosch develops and implements the algorithm for the ACU, together with various Airbag Application tools which are used to ensure its reliability and precision. The crash signals are collected from the automotive companies. Then, these signals are converted into crash data. This crash data contains amplitude values for all sensors over time, and other required data. This data is collected from the customers.

Crash dummies are human like figures which are placed in a car during a crash test. The crash signals of these crash tests are recorded by the customer. Simulations are to be performed on these crash signals. This indicates whether the installed airbags fire at the right time or not. These fire times vary for each vehicle model.

Figure 1.8 shows an example of the crash test performed using a crash dummy. The data obtained from these crash tests are used to determine the activation of passive safety systems. Depending on the detected crash type, crash severity and the respective automotive body characteristics, the passive safety systems are activated by the ACU. [6]

**Figure 1.8: Frontal Car Crash Test [6]**

For example, a frontal crash where the speed of the car was 21 Kmph, collides against a small pole. During this crash scenario, only the belt tensioner is to be activated. In case of a severe frontal crash where the speed of the car is 74 Kmph collides, the airbag which is placed in the steering wheel and dash board are fully inflated. The deployment times of different airbags in various scenarios is determined by the ACU.

Figure 1.9 shows an example of a crash signal. The crash scenario is of a frontal crash, and speed of the vehicle is 24 Kmph. This crash signal is recorded by automotive companies during the lab crash test. [6]

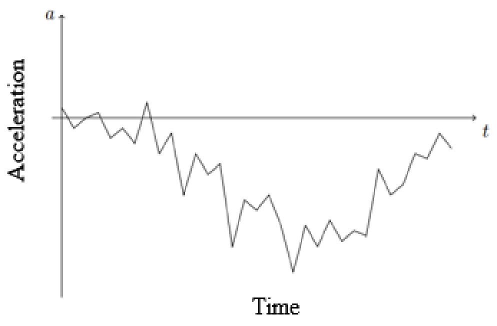


**Figure 1.9: Frontal Crash Test Signal [6]**

The crash data is collected for a specific model of the vehicle equipped with crash dummies and numerous sensors. From this collected crash data, numerous simulations are performed.

The Airbag Application tool named “Simulation tool” converts the crash signal, and then simulates and analyzes the deployment of airbags. The simulation tool uses mathematical models to simulate the behavior of airbags in a virtual environment. This tool models the behavior of airbags under different crash scenarios and helps optimize the design of airbag systems for safety of occupants. Using the simulation tool, engineers can optimize and maximize the performance of airbag systems.

The simulation tool used at Bosch accepts input from the user and produces the resulting output after performing various simulations. The simulation tool simulates crashes while considering assumptions like if the driver is belted or if there are passengers. Depending on these settings, the logic for firing passive safety systems is determined.

For example, if the passenger airbag is deactivated because a baby is placed there, then the airbag should not fire in case of a crash. Passive safety systems have to work correctly under every possible circumstance. These scenarios include, a car driving on a street with potholes. In this scenario, the passive safety system should not be activated.

As depicted in Figure 1.10, the output of the simulation tool is a database file. This database file has the extension as .mdb which stands for “Microsoft Database”. This output database contains large amounts of data such as simulation inputs, simulation results and the airbag fire times for each simulation device. This database contains all the required information for generating the Standard Customer Reports.

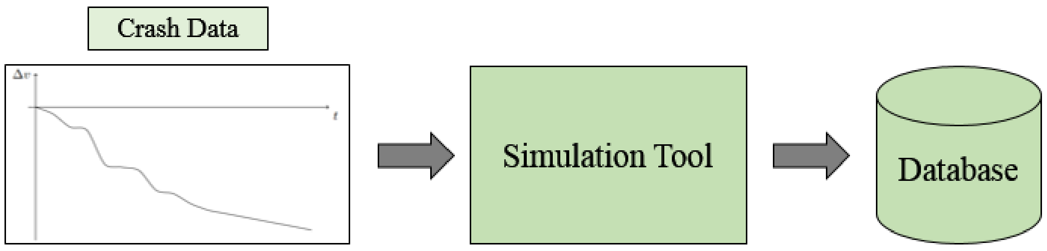


**Figure 1.10: Workflow of the Simulation Tool**

The output of the simulation tool is the result database, which contains simulation information for each type of crash. Each crash is identified with a unique crash ID. Each crash is simulated with different simulation devices and the airbag fire times are determined.

The simulation tool is used to simulate and evaluate complete customer projects for each vehicle model. The exact parameters for each simulated crash are set according to the customer‘s requirements. The threshold functions are then calculated with this data.

To summarise, the airbag systems play a crucial role in protecting the occupant‘s life in case of a crash. During a collision, the airbag inflation should happen at the right time. The firing at the correct time is decided by the Airbag Control Unit. To optimize the functionality of the airbag system, they need to be configured correctly. Simulations are performed in a virtual environment using the simulation tool. Finally, the firing times of these passive safety devices are determined based on the specific vehicle model.

## 1.2 Motivation

The main objective of this thesis work is to generate the Standard Customer Reports for Airbag Simulation Results in an efficient and faster manner.

An airbag application tool should be developed for generating the Standard Customer Reports. The input data to this tool should be extracted from the Airbag Simulation Results. The required calculations are to be performed on the input data. The required values should be stored internally after the calculations. These internal results are then to be stored in the most suitable file format.

The final customer reports have to be developed based on the user requirements. The time taken to generate these reports must be faster than the existing method. The handling of the data in the tool should be in an efficient manner, such that the system doesn’t crash or freeze during the entire report generation process.

The tool chain architecture of all the airbag application tools must be analyzed. The developed tool should then be integrated to the tool chain architecture. It must be able to function along with the current tool chain. Furthermore, this developed tool should not use Excel macros.

Therefore, the main motivation of the thesis work is to generate Standard Customer Reports faster than the previous method. Along with this, the whole stability of the system is to be ensured when using the developed method.

## 1.3 Problem Statement

Generating the Standard Customer Reports using the existing method is complex and uncomfortable to the user. The user must perform several manual steps in order to produce the required reports. The time taken to generate these reports using the existing method is difficult and consumes a lot of time.

The existing method uses Excel macros, which takes a lot of time to generate the required reports. The stability of the computer system is compromised using the existing method. It is also unstable and not user-friendly. The computer freezes and turns unresponsive during the usage of the existing method. The usage of other Microsoft Office products during the runtime of the existing method is not possible. Furthermore, the performance of the overall system is reduced.

When the input dataset is large or if the calculations to be performed are complex, then the method causes performance issues. For example, if the existing method is run in the background, it can consume a lot of system resources without the user being aware of it. This slows down the whole system and becomes unresponsive, making it difficult for the user to work. The existing method cannot be updated as and when the input data changes. Therefore, maintenance of the existing method is difficult and can cause unexpected problems to the user.

The user has to perform several manual steps to obtain the required results. The user has to learn the various steps to use the existing method in order to generate the Standard Customer Reports. This can get quite complex and is also difficult for the user.

These are the drawbacks of utilizing the tool. Due to these problems, the user sometimes has to start the process of generating the Standard Customer Report the previous day and wait for the report the next day even.

## 1.4 Outline of the Report

This thesis report is divided into six chapters, starting from introduction to conclusion. Introduction is the first chapter, which describes about the car crash scenarios and how airbags are used in various scenarios. Airbag systems and the process of obtaining airbag simulation results is discussed in the first chapter. Then, the motivation of the thesis work and the problem statement are covered in this chapter.

The second chapter is the state - of - the - art chapter, which explains the current developments and related research in the field. This chapter provides the existing practices and also identifies research gaps. The concept chapter explains the developed concept, showcasing the possible options for a particular issue. In the concept chapter, handling airbag simulation results, various file formats and the steps to generate output are discussed. Then, after discussing the feasible options, the most suitable concept is chosen for implementation. The implementation chapter contains information on the technical implementation of the developed concept. Input to the developed tool, computations on the input data, and the output report generation is discussed in this chapter. Furthermore, the testing methods used are discussed. In the results and analysis chapter, the final results of this work are discussed, and the results are evaluated. The conclusion and future scope chapter contains the overview of the whole thesis topic and future scope.

# 2 State - of - the - Art

The state - of - the - art chapter discusses about the advancements in the field of airbag systems, along with the methodologies for generating reports. There are four sections in this chapter. The scientific advancements and research in the area of airbag systems are covered in the first section. Airbag systems which can detect different types of occupants and fire the airbag accordingly, is discussed. The second section of state - of - the - art chapter discusses the current trends and developments in the automotive sector. The third section discusses the methodologies which are developed to generate reports. Lastly, the summary section contains the summary of this chapter.

## 2.1 Scientific Developments

In recent years, automotive safety has improved and transformed significantly. The aim is to reduce the number of fatalities and injuries in the event of a car crash. One of the key areas is the effectiveness of airbag safety systems. To enhance their effectiveness, numerous studies and researches have been carried out. With these scientific developments, performance, reliability and safety of airbag systems is improving. Some of the most recent developments in research are described in this section. These research papers highlight the scientific developments in the field of airbag systems.

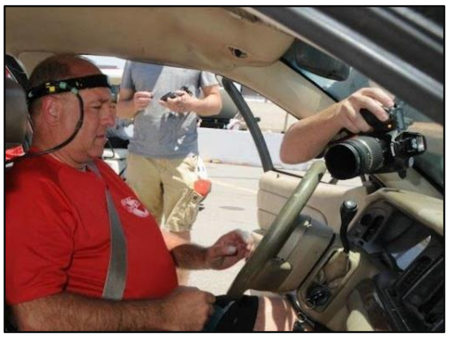
In this research paper [7], authors discussed about the simulation tools used for a particular crash. The crash used is the rollover crash. Occupants' protection during this type of car crash is also described in this paper. Passive safety systems like airbags and belt tensioners have to be activated at the right time in case of a rollover crash. Belt tensioners help the occupant to remain at the same place. During this type of car crash, the airbag which should fire is the curtain airbag. When this airbag is inflated, it cushions the occupant's head and avoids severe injuries.

An advanced airbag system was proposed, to protect the occupant in case of a rollover accident. An experiment was performed for various scenarios. The research distinguishes between different age groups of an occupant.  Based on this, the airbag firing time is decided.

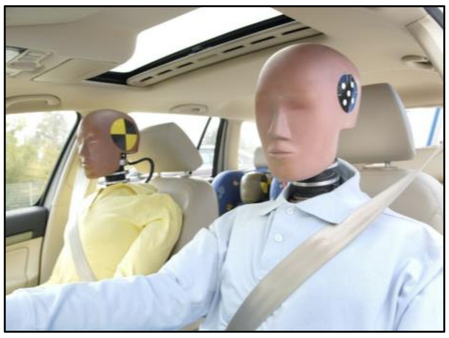
Full scale tests are performed in this research. These tests are performed on two subjects, and they are the human volunteer and crash dummy. This is done to collect data on impact of various crashes on the subject. The human body component that suffers the most damage in a rollover accident is the head. Table 2.1 shows the tests performed on both human and the dummy. Different crash types are used for test specifications. Translation crash is a type of crash where a car is accelerated forward. This car collides with an obstacle. Rotation crash is a type of crash which is similar to a rollover accident. These crash tests are performed with different specifications.

| Test no. | Specification |
|---|---|
| 1 | Pure rotation |
| 2 | Pure translation |
| 3 | Translation, then a following rotation |
| 4 | Pure low translation |
| 5 | Rotation, then a following translation |

**Table 2.1: Test Specifications**



**a) Human Volunteer**



**b) Test Dummy**

**Figure 2.1: Respective Setups to Perform Crash Tests [7]**

In the test setup displayed in Figure 2.1, the human is on the left side. A dummy with similar size and weight is on the right. The weight of the human is 74.5 Kg, and the height is 178 cm. The weight and height of the dummy used for performing the test are 76 Kg and 175 cm respectively. Each of the tests specified in Table 2.1 are performed by one test following the other. These are performed on the human, and then the dummy. During each test, sensors record the displacement of different parts of the body. Figure 2.2 shows a comparison of head displacement for the 5th test specification.

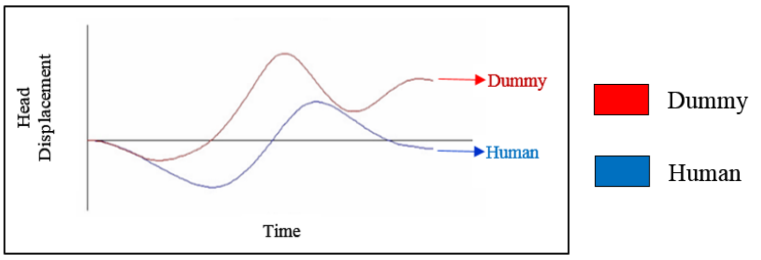


**Figure 2.2: Comparison of Head Displacement between Human and Dummy [7]**

The tests usually performed are mainly by using crash dummies. Crash dummies are used to mimic actual crash scenarios. The data is collected for both human and dummy. This head displacement data which is collected, can be used for the analysis of critical situations. Comparing the actual crash using a human and a dummy can provide even better insights.

Using the collected test data, multiple simulations are performed. These simulations include different speeds, different car size and various ways a car can topple. The output of every simulation performed contains the acceleration, displacement and angle of the vehicle. The results of all simulations are analyzed. Based on this analysis, optimal firing times of airbags and belt tensioners is decided. After looking at this output data, the RTTF is finally agreed upon by the customer and the provider. This is done to provide better protection to occupants in the scenario of a rollover accident.

The firing times of passive safety systems need to be optimized. There are scenarios where a child is placed in the front seat. In this scenario, the child seated in the front seat is belted. During this scenario, the airbag must not fire. In this research, the advanced deployment of the airbag systems was evaluated using event data recorders.

A total of 78 different crash scenarios were created and tests were performed [8]. Few of the test scenarios are as follows:

- **Different Models of a Car:** The vehicle is the same, but different models of the same car is considered for tests. The vehicles were produced in different years. For e.g., for a car which was initially launched in 2019, further models of the same car which are launched in the coming years is used for the tests.

- **Varying Occupant Belt Status:** There are 4 possible cases which are considered for tests. First scenario is a common scenario, where the occupant of the car is belted. The second case is when the occupant is not belted. The third and fourth cases are when the status is unknown and data is incomplete respectively.

- **Different Crash Scenarios:** Frontal crash, side impact crash, roll over crash and even unknown crash scenarios are considered for performing crash tests.

- **Vehicle Body Types:** For performing crash tests, different car types are considered. Sport utility vehicles, also known as SUVs and sedan cars are used for car crash tests.

Under different crash scenarios, the airbag fire results are recorded. The tests are performed on various car models, varying occupant belt status, and various other factors. Advanced airbag deployment functions can be designed and implemented using this data. This helps in reducing the fatalities caused and reduces injuries to occupants.

Advanced airbag deployment systems include airbag suppression. Airbag suppression means that the firing of the airbag system is disabled under various circumstances. In an advanced airbag system, the driver and passenger airbags can operate separately. For a frontal crash, it doesn't mean that both the airbags have to deploy. The airbag must be suppressed when a child is placed in the front seat. So, for varying conditions, the deployment of airbag is to be suppressed.

Advanced airbag systems including airbag suppression is considered for tests. The airbag firing cases for a front passenger side airbag is recorded. This test data is as shown in Table 2.2. During the first scenario, when an adult was situated in the front passenger side, out of 21 crash tests, the airbag is deployed at the right time in twenty scenarios. The airbag is suppressed in one scenario. This suppression happened when the crash severity was less.

In the second scenario, when a child is placed in the passenger seat, the airbag does not fire as shown in Table 2.2. There is one case where a 5 year old child was present in the passenger seat of the car. In this case, the child was not placed inside the crib. The airbag fired at the right time, and advanced airbag suppression also happened. Due to this, the child suffered only a few injuries.

| Right Front Passenger | Airbag Deployed | Airbag Suppressed | Total |
|---|---|---|---|
| Adult | 20 | 1 | 21 |
| Child | - | 1 | 1 |
| None | 14 | 38 | 52 |
| Incomplete data | 2 | 2 | 4 |
| Total | 36 | 42 | 78 |

**Table 2.2: Cases of Airbag Deployment for a Passenger**

The performance of advanced airbag systems is evaluated. The firing of the airbag systems and the belt tensioners at the right time is important to save the occupant's life. The advanced airbag system helps detect various scenarios. Then a decision is made, whether the airbag systems is to be deployed or not.

Around 100 children were killed by airbags during minor car crashes. During these minor crashes, it would have been better if the airbags had not even fired. So, to avoid this problem, many solutions were proposed. By including advanced occupant sensing systems in a car, the airbag firing systems drastically improved. For e.g., occupant sensing systems such as sensors can detect the weight of an occupant. Using cameras, the occupant can be classified as a child or an adult. Based on this classification, the airbag firing can be optimized.

In this research [9], the classification of an occupant is demonstrated. Based on this classification, firing time of the airbag system can be optimized. The research work provided an accuracy of 98% in classifying the occupant as a child or an adult. Along with the occupant classification system, an occupant warning system was also designed. With the help of this warning system, the occupant is warned when close to the airbag firing region. The occupant warning system informs the driver if an occupant is too close to the airbag within 7 milliseconds.

An experiment was performed using a grayscale camera to classify and track the movement of the occupant. The unsafe area where the occupant is too close to the airbag is known as Automatic Suppression Zone (ASZ). If the occupant is in the ASZ, then firing the airbag may cause severe injuries to the occupant. **S**uppressing airbags depending on the type of occupant is performed. The possible mechanisms for suppressing airbags is as shown in Figure 2.3.

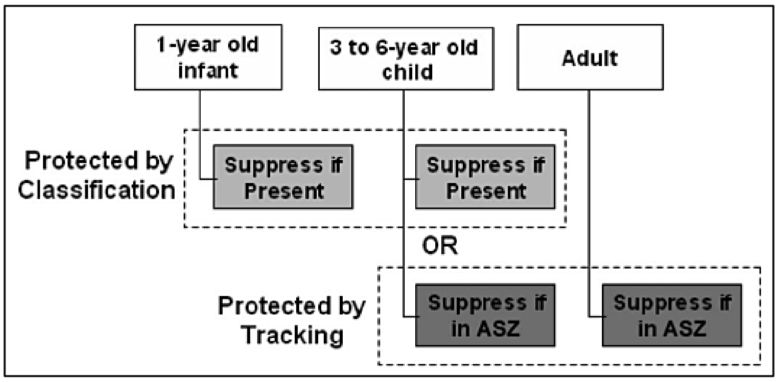


**Figure 2.3: Cases for Airbag Suppression [9]**

The camera zone is also dependent for tracking and classification mechanisms. The classification and tracking mechanism improves the functionality of airbag systems. Therefore, this proposed methodology can warn the occupant and save their lives. Along with the proposed methodology, airbag systems to protect pedestrians in case of a car accident are examined.
Lives of pedestrians may be lost when they are involved in a car accident. To avoid this, pedestrian protection systems were invented. During a car collision with a pedestrian, the

person’s legs are usually the first point of impact. The next impact is with the hip region. Lastly, the head of the pedestrian may hit the hood or front glass window. Sensors installed in a car are the first to detect the collision with a pedestrian. A sophisticated algorithm calculates to know whether the object hit the car or not, and the type of object. Shortly the assigned protection systems are activated. This protection system is called the active pedestrian protection system. It is triggered when it receives a signal from the sensor. The pedestrian airbags are then fired. The lives of pedestrian’s can be saved by these airbags.

The latest advancements and scientific developments in the field of airbag systems is discussed in this section. These improvements optimize the firing of the airbag systems. They save the lives of occupants and avoid severe injuries.

## 2.2 Current Trends

This section discusses the latest trends in the automotive sector. The latest developments by the Computer Engineering Department at the Chemnitz University of Technology are discussed.

The traditional software development for vehicles involves the use of V- model. This development of software happens in a sequential process. The V- model development process cycle is as shown in Figure 2.4. The software development starts from requirements, and then is the specification phase. The final step is acceptance testing.

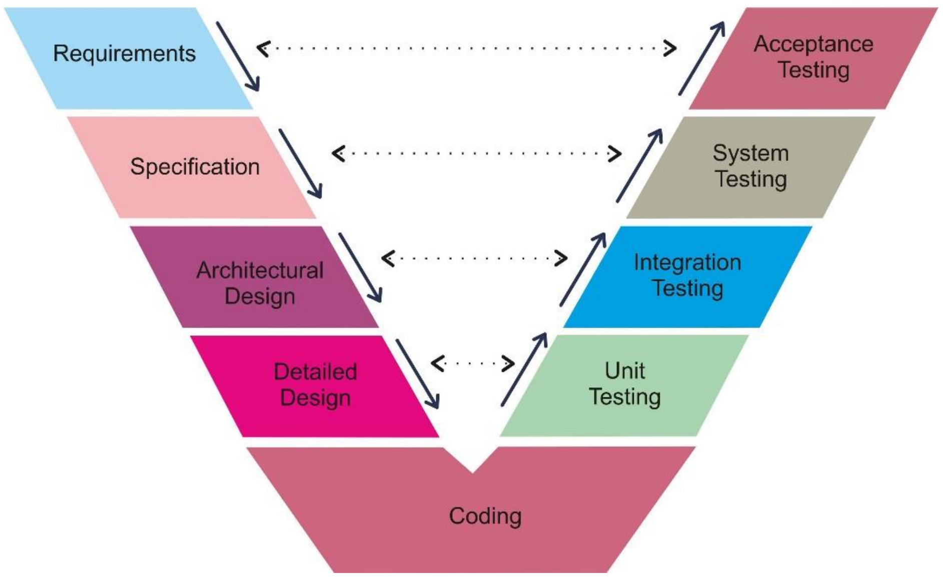


**Figure 2.4: V- Model Development Process [10]**

The demands of the automotive industry are increasing. To meet these demands, an agile software development process is proposed. This agile development process has continuous integration of the software components to the software suppliers. The main target of this process is to develop a model, and then improve it in the subsequent deliveries. The continuous integration (CI) is a method, where software developers can directly integrate their work to the central source repository on an everyday basis. Testing is performed at every step before code integration. [11]

The V- model development process has many disadvantages, such as limited flexibility and less adaptability to change. So, the agile development process is proposed in this research. The following are the main advantages of the proposed methodology:

- Detection of errors is easier
- Frequent release of latest updates
- Integration is automated and faster
- Developers can continue without waiting for testing
- The overall system is not disrupted

So, the continuous integration of developed software to the server can be used for better software development. Tools such as GitHub or IBM Rational Team Concert (RTC), can be used to meet the industry demands. These tools are used for version control, source repository and tracking work changes. Automotive Open System Architecture stands for AUTOSAR, is a simplified layered architecture. AUTOSAR is used to manage the increasing complexity of the software. The main motivation to develop AUTOSAR is as follows:

- To increase the reusablity of application software
- Facilitate the collaboration between different OEMs and suppliers
- It enables easier exchange of the architecture model
- It provides a standard architecture for software development

Providing a standardized methodology like AUTOSAR in the automotive industry can help manage complex software. The AUTOSAR standard is divided into three main layers. These three different abstraction layers in the AUTOSAR architecture is depicted in Figure 2.5. They communicate through interfaces defined by AUTOSAR.

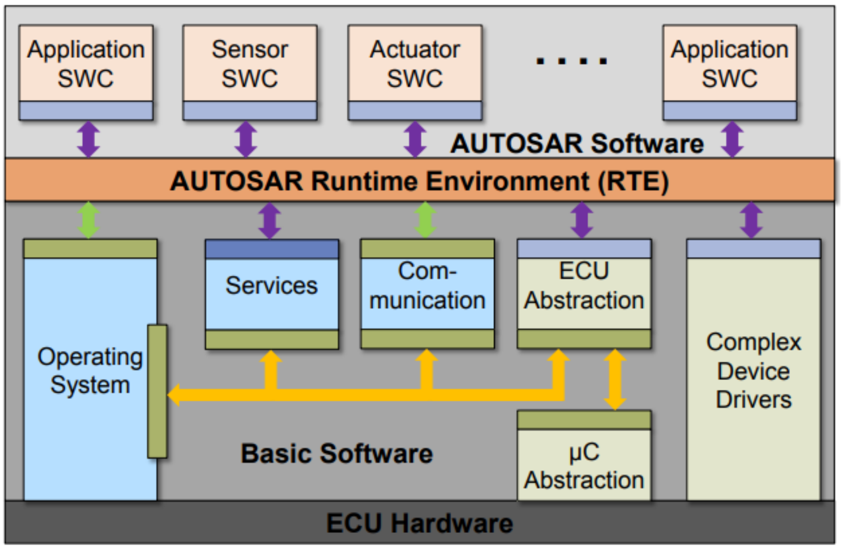


**Figure 2.5: AUTOSAR Software Architecture [12]**

The three layers are application layer, RTE which stands for RunTime Environment, and BSW layer, which is Basic Software layer. The three layers of the AUTOSAR architecture are as follows:

- **Application Layer:** The software components that implement the cars functions and services are handled in this layer.

- **RTE Layer:** This layer provides an environment for execution of the software components.

- **Basic Software Layer:** An infrastructure is provided for the application layer by the BSW layer. This layer also provides required services such as memory, input/output, and communication management. [12]

Each layer of the standard AUTOSAR software architecture is briefly described. This AUTOSAR architecture is designed to be independent of the platform used. This implies that it may be used on different platforms. Hence, the developed software component can be resued. This reduces the development costs and time. This standard approach enables development of high quality software which is maintainable easily and reliable. Furthermore, AUTOSAR architecture provides a standard approach to develop software for automotives.

The Computer Engineering Department at the Chemnitz University of Technology is currently involved in a number of ongoing research projects [14]. It aims at advancing the field of computer engineering, and developing innovative solutions.

One of the ongoing projects at the department is the YellowCar project. The YellowCar is an automotive vehicle demonstrator platform. This platform can be used for performing functional test, performance evaluation and optimization of hardware. It has Electronic Control Units (ECUs), and they use AUTOSAR standards.

The YellowCar contains 3 ECUs, which are the assistant ECU, processing ECU, and feature ECU. The three ECUs are connected to the sever through a CAN (Controller Area Network) bus system as shown in Figure 2.6. [13]

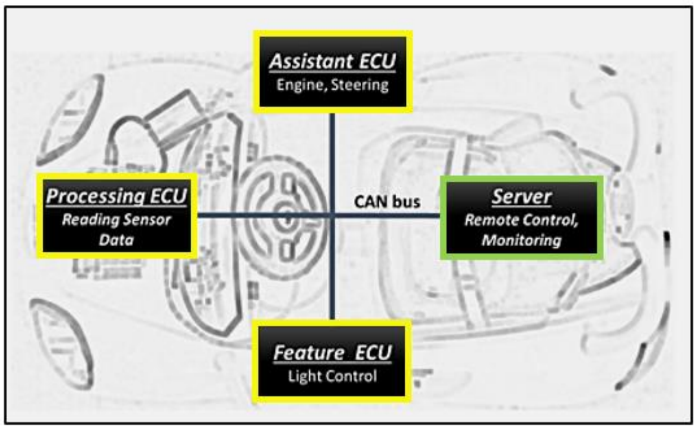


**Figure 2.6: YellowCar ECUs and Server [13]**

The server is implemented in C++ programming language. The server provides a webpage representation of all the sensor data values. These sensor values are received by sensors on the YellowCar through the CAN bus.

A client running the simulation of YellowCar receives data from the YellowCar server using C# software. This simulation of the YellowCar represents a 3-D model, which can perform the same actions like a real car.

The department further developed an extended model of the YellowCar. It is named as the BlackPearl. The BlackPearl automotive demonstrator is shown in the Figure 2.7. The BlackPearl contains an extended platform located on top of the vehicle. This extended platform contains a multi-board rack with additional sensors and cameras. This automotive demonstrator called BlackPearl, is mainly used for the implementation of Advanced Driver Assistance Systems (ADAS), image processing tasks, testing of developed modules, and performance evaluation.

**Figure 2.7: BlackPearl Automotive Demonstrator [14]**

The BlackPearl can be installed with up to six ECUs. The ECUs communicate through a CAN bus. With multiple ECUs communicating with one another and performing tasks in parallel, the speed and efficiency of the overall system is enhanced. The computed information is then sent into the the CAN bus, as a CAN message to the server. The server then implements the ADAS functions. They are used for image processing tasks such as image pre-processing, feature point detection and recognizing the target [14].

The sensor unit handles all the inputs from the sensors, including steering control sensors, light control sensors and distance sensors. The installed sensor unit can activate various motors and turn the LEDs on or off. This platform can be used with any simulation and modeling tool, such as Matlab-Simulink. The BlackPearl platform can perform image processing applications for high way traffic analysis. The detection of traffic on a high way can be seen in Figure 2.8. Gaussian filtering is used to reduce unnecessary information and also the complexity. Blob detection and SURF detection algorithms are used for detecting vehicles and lane markings in a high way.

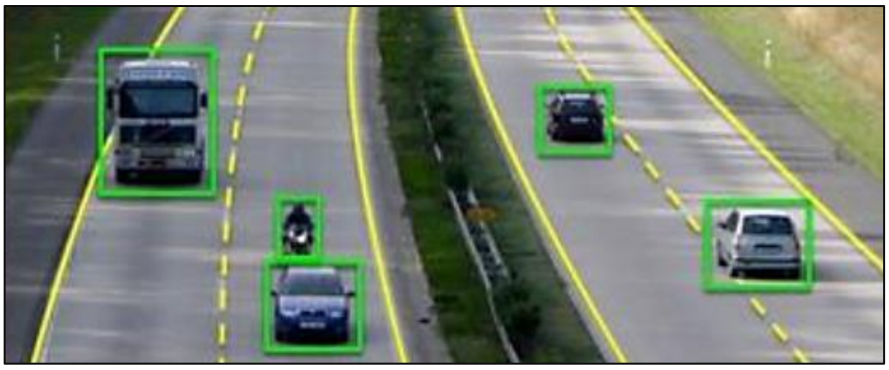


**Figure 2.8: High Way Traffic Analysis [14]**

The BlackPearl automotive demonstrator is also implemented to perform Car2X-Indoor-Lab, evaluate Car2X communication, and automated driving based on cloud computing techniques. The platform also provides functionalities such as traffic sign detection and recognition, light control and Car2X data exchange with the traffic cloud.

This paper descibes the BlackPearl platform and the use of multicore processors to provide high computational power for parallel computing. This paper also explains how the AUTOSAR standards are used for software architecture, implementation, and testing of ADAS applications on the platform.

YellowCar and BlackPearl are built following the standards set by AUTOSAR. Software testing is done to assure its quality. Performing tests at different stages also reduces the development costs and time. The testing process starts with the initiation of an AUTOSAR project. This step involves defining the system architecture, compiling and building code. To ensure that this code compiles with AUTOSAR standards, static testing is performed. In the static testing process, three different static tests are performed. This workflow for starting an AUTOSAR system test can be seen in Figure 2.9. The static tests are performed on three layers of standard architecture. They are as follows:

- **Static Test Application:** This testing is performed to identify any defects in application software code. Tests are performed to verify that the components have been developed according to AUTOSAR standards.
- **Static Test RTE:** Tests are performed on the Run-Time Environment. This verifies the communication of different software components in the system, such that there are no errors. As a result, the system's performance remains unaffected.
- **Static Test BSW:** The basic code is analyzed without executing it. Potential defects or errors are identified during this phase of static testing.

These are the three static tests which can be performed. During the tests, if any defects are identified, then they are resolved. It also ensures that the development cost and time is lowered.

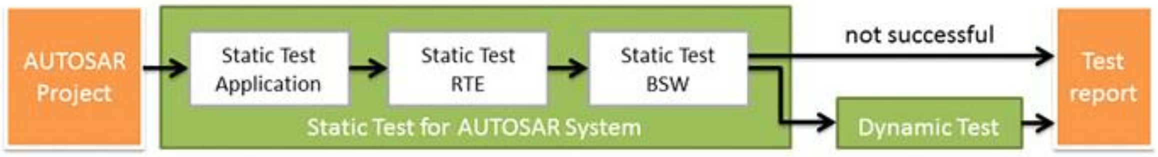


**Figure 2.9: Workflow for AUTOSAR System Tests Approach [15]**

The process begins from starting an AUTOSAR project. Then, the static tests analyze the test object. These static tests do not execute and perform tests. The tests which are performed is independent of the tool used. The test cases can be changed based on the requirements, and based on the database used. Configuration files and source files are checked to ensure high quality software [15]. If the static tests fail, then a test report is generated. This report states that the test cases failed, and static tests are not successful. In this case, the test report contains the details of failed tests. If the static tests are completed successfully, then a test report is generated after performing the tests.

Instead of performing three different static tests, application, RTE and BSW test, another approach with a new tool was proposed. This tool can be seen in Figure 2.10. This unified single static test proposed can perform static testing with the use of a single static test. [15], [16]

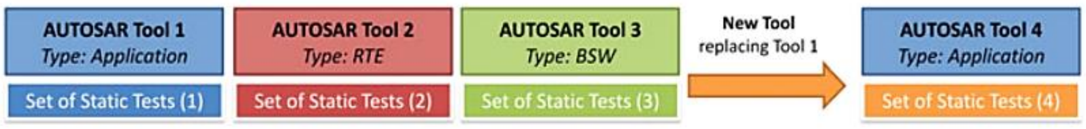


**Figure 2.10: Proposed Approach for Static Tests [15]**

Static tests in the unified method perform checks if there are any illegal direct calls of other functions or methods. This is performed by parsing the file, and then checking the names from the knowledge base to the source file. Test reports are generated in the proposed method for successful and failed tests accordingly. In the proposed method, all of these steps happen in only one phase. There are no additional steps required for performing static tests. The proposed method in Figure 2.10 having only one phase is better than the method having three separate testing phases. In both the methods, reports are generated when the tests are performed. In the proposed methodology, the report generation is done in one phase. In the other approach, the report was generated at the end of three phases. Therefore, the proposed concept to generate reports is better.

Report generation is one of the important steps to analyze data and present information in a precise and clear manner. In the next section, various methodologies to generate reports are discussed.

## 2.3 Methodologies for Report Generation

This section of the chapter discusses various methodologies which are used to generate reports. These practices are typically followed where report generation is necessary.
An approach of including configuration files to the source code to perform static tests is proposed. This approach is used to develop high quality AUTOSAR software, and also reuse developed components efficiently. The configuration files contain information about the AUTOSAR ECU structure, function names and parameters. This input information is used to perform the required static tests. After performing the required computations and validation of the source code, output reports are generated.

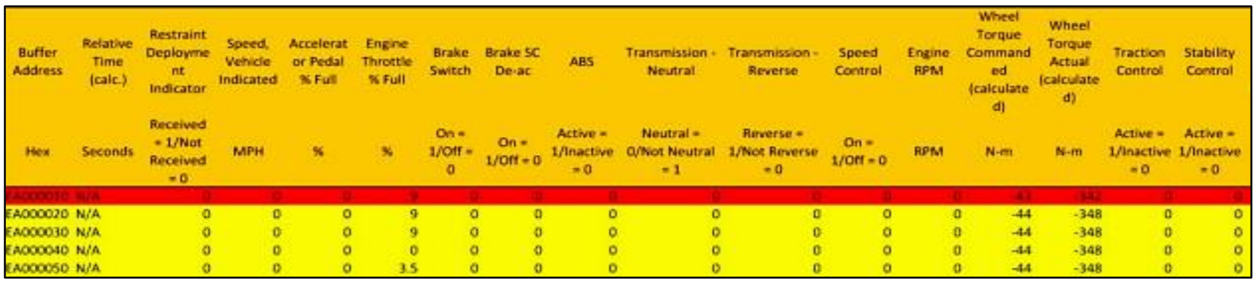
Along with the test reports, the results are displayed in a GUI, which stands for Graphical User Interface. Flags are used to represent any missing or incomplete modules. These flags help testers and developers find solutions. Furthermore, the proposed methodology uses ARXML which stands for AUTOSAR XML. A knowledge base is created in the end, which consists of the interfaces and their associated parameters. This knowledge base is created by parsing all the existing folders in the project, including the input files. [17], [18]

In this research, a tool named CDR (Crash Data Retrieval), is used by the customers to extract data from the powertrain control modules. Using this tool, customers were able to safely extract data. They are as follows:

- Airbag system status
- Speed of the vehicle
- Brake status
- Crash related data

These are some of the important data which is recorded and can be extracted. The data can contain upto 16 different parameters based on customer requirements. This information is recorded as soon as the car is turned on, and ends as soon as the car is turned off. Any crash related data is recorded during that time. [19]

The process of generating reports from this data is a tedious process for the customers. It can seem like it is a simple copy paste from the input data to an Excel spreadsheet. This task becomes much more complicated for varying input data. Example of the report generated by the tool can be seen in Figure 2.11.



| Buffer Address | Relative Time (calc.) | Restraint Deployment Indicator | Speed, Vehicle Indicated | Accelerator Pedal % Full | Engine Throttle % Full | Brake Switch | Brake SC De-ac | ABS | Transmission - Neutral | Transmission - Reverse | Speed Control | Engine RPM | Wheel Torque Commanded (calculated) | Wheel Torque Actual (calculated) | Traction Control | Stability Control |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hex | Seconds | Received = 1/Not Received = 0 | MPH | % | % | On = 1/Off = 0 | On = 1/Off = 0 | Active = 1/Inactive = 0 | Neutral = 0/Not Neutral = 1 | Reverse = 1/Not Reverse = 0 | On = 1/Off = 0 | RPM | N-m | N-m | Active = 1/Inactive = 0 | Active = 1/Inactive = 0 |
| EA000010 | N/A | 0 | 0 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -43 | -342 | 0 | 0 |
| EA000020 | N/A | 0 | 0 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -44 | -348 | 0 | 0 |
| EA000030 | N/A | 0 | 0 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -44 | -348 | 0 | 0 |
| EA000040 | N/A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -44 | -348 | 0 | 0 |
| EA000050 | N/A | 0 | 0 | 0 | 3.5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -44 | -348 | 0 | 0 |

**Figure 2.11: Example of a Report Generated by CDR Tool [19]**

The user must organize and format the data in an Excel spreadsheet. The manual report generation is error prone and can be time consuming. Hence, this report generation method was quite difficult.

This research introduced an approach to represent the VHDL-AMS models. AMS in VHDL-AMS stands for Analog and Mixed Signal. This model is created to simulate both analog and digital circuits in a single environment. For the VHDL-AMS model representation, XML (eXtensible Markup Language) which is the representation was selected as an intermediary step. The integration of tools such as simulators, design checkers and debuggers can be based on this standardized interface. [20]

The element structure and the element content are defined in the intermediate step. Finally, the design structures were exported to HTML (Hyper Text Markup Language). This approach produces multiple outputs from the intermediate files. To convert from the intermediate files to the output files, two possible options are discussed.

The first option is XML Script language. The second option is XSL Transformations (eXtensible StyleSheet Language). As the XML data for the project was less complex, the first option was selected. For this particular approach, XML Scripts were more suitable. [20]

This method contains three main scripts which are known as templates. The templates which are responsible for generating the final report are:

- ➢ First script for traversing, also called the major template
- ➢ Second script for creating a navigatable tree structure
- ➢ Third script for creating a page with statistics

These are the templates which are used for report generation. Along with these three scripts, a number of sub-templates are used for the transformation from one format to the final output. For visualization of the elaborated VHDL-AMS data structure, this approach was proposed.

In this research paper, a report generation method is proposed using JSON (JavaScript Object Notation) file format. This research compares both XML and JSON file formats for storing intermediate data. Finally, JSON is used as the intermediate file format type. Based on the input data available for this project, JSON file format was suitable. The report generated using the intermediate file is a HTML file. This output report can be seen by the user. The process to generate the reports involves multiple steps [21], [22].

The research paper to generate reports uses VBA (Visual Basic for Applications). This approach has associated VBA code. This programming language is integrated with Excel. It can perform repetitive tasks. For e.g., VBA code can generate reports by doing some set of tasks repeatedly. The resulting report is exported as a Microsoft Excel file or a PDF document in this method. Reports can be created easily when the data set being used is simple. However, as the size of the data grows, the process becomes complicated and slow [23].

The methodologies used to generate reports are summarized in Table 2.3. This table contains the representation of data, and the conversion methodologies used to get the final output report. Disadvantages of the discussed methodologies are elaborated further.

| **Research Paper** | **Internal Representation** | **External Representation** | **Conversion** | **Final Output** |
|---|---|---|---|---|
| [1] | C | *ARXML* | *JavaScript* | HTML |
| [2] | Text file | - | *Manual* | Excel |
| [3] | Java | *XML* | *XML Script* | HTML |
| [4] | C | *JSON* | *JavaScript* | HTML |
| [5] | Excel | - | *VBA* | Excel/PDF |

**Table 2.3: Different Methodologies to Generate Reports**

The disadvantages of the methods are as follows:

- The arrow (→) represents the next step or the next file which is generated. This is mainly from the first column (Internal Representation) in Table 2.3 to the last column (Final Output).

- The plus symbol (+) represents the way conversion takes place. This is basically the two columns in between, which is External Representation and Conversion.

[1] C $\longrightarrow$ *ARXML* + *JavaScript* $\longrightarrow$ HTML

- The conversion process between C and ARXML is time consuming, and sometimes error prone.
- HTML output is not suitable for reports, as it requires additional styling.

[2] Text file $\longrightarrow$ *Manual Conversion* $\longrightarrow$ Excel

- ➢ Manual conversion of data is time consuming
- ➢ If the input data is large, then a lot human effort is required
- ➢ Human errors can occur

[3] Java $\longrightarrow$ *XML* + *XML Script* $\longrightarrow$ HTML

- ➢ XML Scripts are not ideal for large scale applications
- ➢ HTML output reports are not visually appealing
- ➢ XML Script provides limited support and functionality
- ➢ Difficult to maintain the written code

[4] C $\longrightarrow$ *JSON* + *JavaScript* $\longrightarrow$ HTML

- ➢ JSON supports only limited data types
- ➢ In JSON, data is not validated. This means that the data being passed is not checked for the correct datatype. [22]
- ➢ Can lead to errors and inconsistencies
- ➢ JavaScript has performance issues

[5] Excel $\longrightarrow$ *VBA* $\longrightarrow$ Excel / PDF

- ➢ VBA code is difficult to maintain and update
- ➢ Functionality of VBA code is limited to perform repetitive tasks only
- ➢ A lot of time is taken to generate the output reports

This section discusses the current methodologies used to generate reports and the existing research gap. The existing approaches have their set of advantages for the particular use case in the research paper. They also have their disadvantages for other use cases. To generate reports, an efficient methodology is required. The efficiency of each methodology can differ. This is based on the particular use case, and their set of requirements.

## 2.4 Summary

The first section of this chapter consists of the scientific developments of airbag systems. Recent developments include advanced airbag systems. These systems can consider the occupant’s size and age to determine the airbag firing time. This is discussed in detail in this section. Airbag technology is growing continuously with the latest developments and research.

In the second section of this chapter, current trends in the software development process, along with the use of AUTOSAR methodology for software architecture is discussed. This section also explores the latest developments in the Computer Engineering Department at Chemnitz University of Technology. The automotive demonstrator platforms, the importance and ways of static testing is discussed. Furthermore, the generation of output reports is also briefly explained.

Lastly, the third section discusses the existing methodologies for generating reports. These methodologies involve using different representation of data. Along with this data, conversion techniques to generate the final output is presented. This section highlights the selection of the appropriate methodology for a particular use case. After analyzing the existing techniques, the drawbacks of these techniques are highlighted in this section.

# 3 Concept

In this chapter, the proposed methodology for generating Standard Customer Reports is discussed. This chapter comprises of are three major sections. The first section presents the existing methodology which is used to generate Standard Customer Reports. The second section demonstrates the proposed approach, beginning from handling the simulation results to generating the required reports. The second section of this chapter is further sub-divided into three parts. In the first part, handling of airbag simulation results is discussed in detail. In the second part, the possible intermediate file formats are examined, and the most suitable file format is chosen. The final report generation is described in the third section. Lastly, the third section contains the summary of this chapter.

## 3.1 Existing Concept

The existing methodology to generate Standard Customer Reports is discussed. This methodology involves extracting airbag simulation results, and processing it into a report. This report must be easily readable by the user and the customer. The existing method to generate the reports is further analyzed. The methodology's disadvantages are then discussed.

The generation of reports with the existing methodology uses Excel macros. Excel macros are used to perform repetitive tasks in Microsoft Excel. One possibility to create a simple macro is by starting a feature named macro recorder. When the recording has been started, the developer performs several mouse clicks and keystrokes. After performing the clicks, the recording needs to be stopped. Then, an Excel macro is created. This macro can be used in other Excel files to perform the required operations. Another possibility to create complex macros is with Excel VBA. This is the way in which macros are created in the existing concept.

For e.g., an Excel file contains the names of crashes such as frontal crash and rear crash. A macro can be created in this Excel file. This macro is created to highlight frontal crash with red background color. This macro will then be saved. Another Excel file with similar data, containing the crash names is opened. The created macro is run in the opened Excel file. Then, all the frontal crashes will be highlighted with red color in this file. This is a simple example of the functionality of an Excel macro.

Using the existing methodology to create the required Standard Customer Report, the user needs to perform many manual steps. Firstly, an Excel macro which is created for this purpose needs to be installed on the user's computer. The user has to perform multiple clicks in the Excel file, in a specific order. These multiple clicks which have to be done by the user are shown in Figure 3.1. After performing all of these clicks, the macro is started. These clicks are necessary to prepare the airbag simulation results and then generate the required reports. The order in which the clicks have to be performed are as follows:

1) Click on the "Add-ins" tab
2) Then, "MDS-Macros"
3) "3-Crashes"
4) "Step 1"
5) "MDS"
6) "Default"

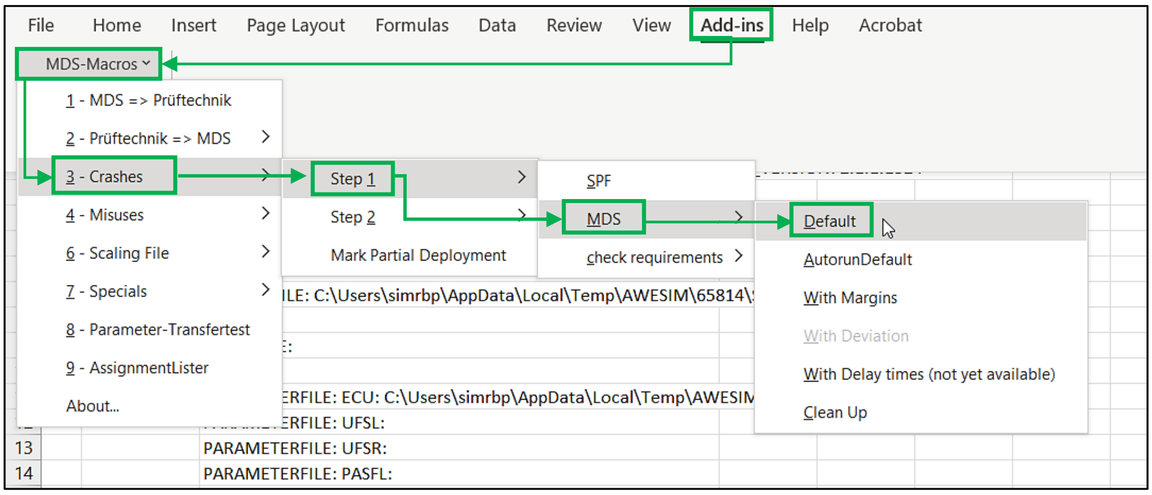
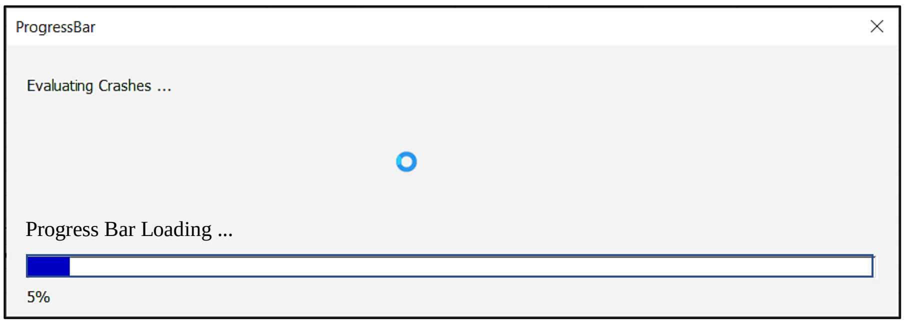


**Figure 3.1: Steps to Run the Excel Macro**

When the macro is started, the Excel application displays a progress bar in a window. This window displays the status of the task. An example of the progress bar which is displayed during the usage of the macro can be seen in Figure 3.2. First, the crashes are evaluated. After the completion of the progress bar for the first time, this progress bar starts again. The reason for the progress bar starting again is to format the data. The progress bar displays the percentage of completion for formatting the data. After the second progress bar completes, the Standard Customer Report is generated.


ProgressBar

Evaluating Crashes ...

Progress Bar Loading ...

5%

**Figure 3.2: Progress Bar after Running the Macro**

Using these steps, the user can generate Standard Crash Reports. There are some disadvantages when Excel macros are used to generate the crash reports.

Time Consumption: One of the main negative effects of using Excel macros is that a lot of time is taken to generate the required reports. As the number of repetitive tasks to be performed by the macro increases, the time taken to generate reports also increases exponentially. If the quality of the developed macro is not good, then the time factor taken to generate reports is quite high.

System Instability: System instability is a significant issue while generating Standard Customer Reports using Excel macros. The system often freezes and becomes unresponsive. Along with this, other Microsoft Office products cannot be used during report generation. If there are errors such as infinite loops in the macro, then the system slows down. This can even lead to an unexpected system crash, where the report is not generated. Due to the instability caused during report generation by using Excel macros, the performance of the system is reduced.

The macro package needs to be installed. Then, the macro can be started with some manual mouse clicks. This extra macro step can be avoided. A macro can be updated or modified as required, but the process to do this may be quite complex.

The disadvantages of using Excel macros, such as time consumption and system instability makes it complex for the user to generate reports. The additional step can be avoided. Therefore, the report generation methodology used to generate Standard Crash Reports needs to be replaced. A new methodology is proposed further. The proposed approach must overcome the drawbacks of using the existing methodology.

## 3.2 Proposed Concept to Generate Standard Customer Reports

This section presents a methodology to generate Standard Customer Reports. The existing methodology using Excel macros is time consuming, unstable, and requires an unnecessary additional step. The proposed solution must overcome the drawbacks of the existing technique. The proposed approach is discussed further.

Customers perform crash tests by using crash dummies and record the crash data. The crash related data is collected from customers. This data is simulated using the simulation tool. The output from the simulation tool is the airbag simulation result. These simulation results are stored in a database. This is the workflow of the simulation tool. The airbag simulation result is used for generating the required reports. These reports contain airbag deployment times under various simulated scenarios. The airbag behavior can be seen through these reports. These reports are called the Standard Customer Reports.

To generate these reports, a methodology is proposed. This methodology is further explained in the three sub-sections. The possible methods to extract airbag simulation results are presented, and a suitable method is chosen. The file formats to store the simulation results are discussed. Then, the concept of generating the Standard Customer Report is presented.

### *3.2.1* Handling of Airbag Simulation Results

In this sub-section, the concept to extract the required information from the airbag simulation results is discussed. The information is extracted from these results to generate the Standard Customer Reports. Various possible options for getting this input data are discussed, and a suitable option is chosen. The advantages of the programming language and IDE used are discussed. The concept to extract data, and then to store it internally is presented in this sub-section.

The simulation tool creates a database file with all the simulation values of the crash data. Airbag simulation results contain crash-related data such as crash code, crash information, and variation. This data is stored for multiple crashes, and for various simulation devices used. A brief description of the required crash related data for the Standard Crash Report can be seen in Table 3.1.

| Crash Data | Description |
|---|---|
| Crash Code | A unique crash code is assigned to each vehicle crash |
| Crash Information | Type of the crash, and speed of the vehicle are added as additional crash information |
| Variation | Different variations of the crash signal, such as time variation and amplitude variation |
| Simulation Device | Actuator that can be triggered during a crash<br>For e.g., AB1FD - Airbag of the Front Driver |
| Tmin | Minimum fire time of an airbag during a car crash |
| Tmax | Maximum fire time of an airbag during a car crash |
| Rmin | Minimum roll angle (only for roll-over crashes) |
| Rmax | Maximum roll angle (only for roll-over crashes) |
| Deployment rate | The percentage of airbag deployment<br>For e.g., If it is 100%, it means that the airbag has deployed for all time variations |

**Table 3.1: Required Crash Data from Airbag Simulation Results**

This simulated crash results are stored in the crash database. This database is named as RSDB (Result Database). The simulation tool simulates all crashes and the output information is stored in RSDB. An airbag application tool named RSDBnext extracts the required information from the RSDB. RSDBnext tool requires an input database (RSDB).

Along with the database input, the tool also requires another input. The other input is of "Format" type. This type is provided as input to generate different type of reports. These reports contain various crash related information. There are four possible format options which can be provided to the tool. These options generate four different result files.

The possible format types which can be given to RSDBnext are as follows:

- "Result" - This format option is designed to create result files.
- "Scaling" - This format option creates a scaling file. The scaling file contains single results for every time variation.
- "Misuse" - This format option is used to create a misuse report. This report shows the amplitude up to which a stable no fire is seen for a misuse event. No fire is when the airbag does not deploy for a crash.

- “RSDBplus” - This format option creates a result with limited features.

These are the format options which currently exist for the RSDBnext tool. The RSDBnext tool can be started with any of the format options. Depending on the format type, the RSDBnext generates the required output files.

When the format option is provided as “Result”, the RSDBnext tool is started to create the result files. The tool generates an Excel file and an XML file as output. The output XML file contains all the required crash data necessary to generate the Standard Crash Report. The required crash data (which is shown in Table 3.1) needs to be extracted for report generation.

This extraction of the required data from airbag simulation results can be done in two possible ways. These ways are marked as “Option 1” and “Option 2” in Figure 3.3.

1) The first way to extract the required crash data is from the database (RSDB)
2) The second way to extract data is from the XML output of the RSDBnext tool

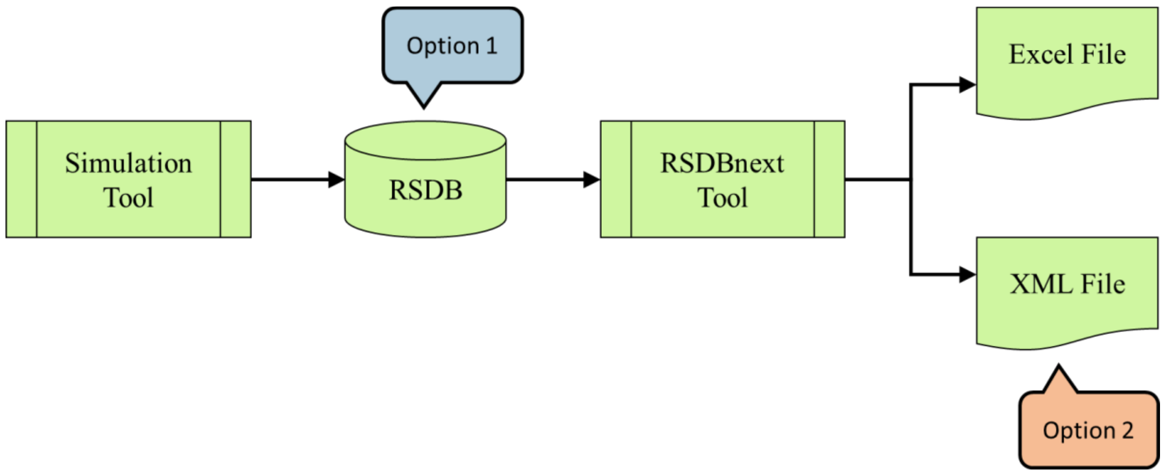


**Figure 3.3: Possible Options to Extract Airbag Simulation Results**

The first option may be quite complex. This method involves retrieving data from a database, which might degrade system's performance. The second option is to extract this data from the output of the RSDBnext tool. RSDBnext tool presents the crash data in an XML format. The crash data is saved in a hierarchical structure in the XML file format. This is easier to parse and retrieve the required crash data. This XML file already contains all the necessary information for generating Standard Customer Reports. Therefore, an extra tool which is doing the same function as the RSDBnext tool is not necessary. So, the second option, which is using the data extraction tool (RSDBnext) is better. Therefore, the second option was chosen for the report generation process as it is better than the first option.

RSDBnext tool reads the database and generates an XML file. This XML file (named as result XML) contains all the required information for the Standard Customer Report. To extract the required crash data, RSDBnext is started with the input format as “Result”.

To generate the crash report, crash data must be extracted and computations are to be performed on it. The data which is extracted should be saved in an internal data structure. On the retrieved data, additional computations are performed.

As a first step, to extract the data and store it in an internal data structure, C# programming language is selected. It is an object oriented language. C# programming language supports the development of complex and modular systems efficiently. This programming language is compatible on various platforms such as Windows, Mac OS and Linux [24]. It has better memory management system compared to other languages.

This language is combined with Microsoft VS 2019. This is an advanced powerful IDE, which provides robust libraries and frameworks. Visual Studio also includes a powerful code editor. This can perform tasks such as automated code formatting and code completion. Using these features, the developed code can be clean and efficient. Developing the tool using C# helps in the integration of the tool into the toolchain. This is because some other tools are also developed in the same language. So, C# is the programming language being used, along with the IDE as Visual Studio 2019.

There are two possible methods to extract data using C# and Visual Studio. This extracted data needs to be saved into an internal structure. The two methods possible to do this are as follows:

1) To develop a new tool altogether
2) To adapt, and further develop the existing RSDBnext tool

Selecting the first option complicates the report generation process for the user. The user must install the new tool, and then start the process. The user must also learn the functionality of the new tool. For e.g., the input for the new tool may be different. It requires more effort to learn about the new functionality. This complicates the report generating process, and is not always user-friendly. Choosing the second option of adapting and further developing the RSDBnext tool is more efficient than developing a new tool altogether. The user knows to use the existing RSDBnext tool. Additional effort and time is not required when the second option is chosen. Considering all these factors, the second option is better than the first option.

The existing tool, RSDBnext is further developed to generate Standard Customer Reports. This must be performed without disturbing the functionality of the existing tool. To generate these reports, the user should start the RSDBnext tool, and provide the format option as “Standard” for Standard Customer Report generation. First, the tool runs with the option as “Result”. Then, the tool runs with the “Standard” option. This workflow can be seen in Figure 3.4.

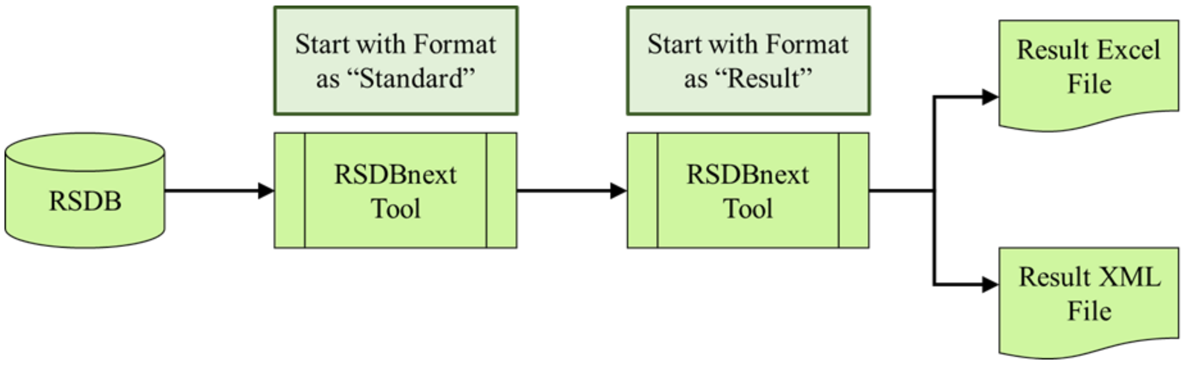


**Figure 3.4: Proposed Approach to Develop RSDBnext Tool**

When the user provides format option as “Standard” and starts the tool, the tool first internally runs with format option as “Result”. The output of the format option “Result” is the result XML file. After the result XML file is generated by the tool, the tool continues with the format option as “Standard”. This result XML file is taken as input for further steps. Using this input, the tool starts generating the required Standard Crash Report. The Standard Customer Report is also known as the Standard Crash Report.

To summarize these steps, the user starts the existing tool RSDBnext with the format option as “Standard”. Then, the tool internally starts RSDBnext with format option as “Result” first. After the RSDBnext tool finishes running for “Result” format option, the result XML file is generated. This will then be used for generating the Standard Crash Report, using “Standard” option.

Once the result XML file is generated, the next step is to extract crash data from it. This result XML file contains crash data in a hierarchical format. The hierarchy of crash data for one crash can be seen in Figure 3.5. This hierarchy is further explained.

The XML file contains data for multiple crashes. Each crash is identified with a unique crash code. Then, each crash is simulated under multiple variations such as amplitude and time variation. Furthermore, each variation is performed for multiple simulation devices. The airbag deployment related values such as Tmin, Tmax, and deployment rate are stored in this level. The extraction of airbag deployment information needs to be done. This extracted information should be stored internally in the tool. Once the required crash data is extracted and stored internally, the computations are then performed on this data.

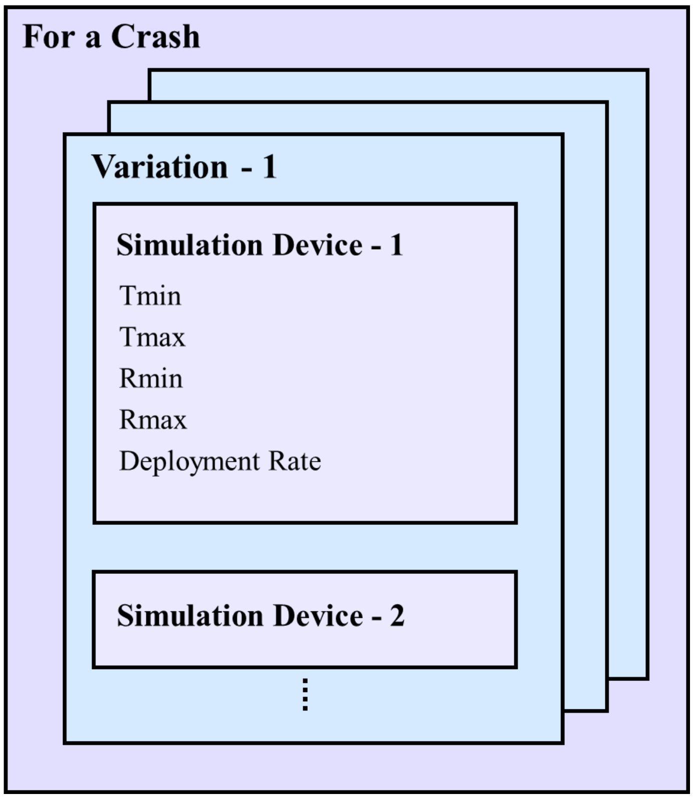


**Figure 3.5: Hierarchy of the Crash Data**

After the hierarchy of the crash data is analyzed, the extraction is done by reading the XML Result file. The XML result file is parsed for multiple crashes in the file. All the require data is then stored internally in an internal data structure. This internal data structure contains all the values required for computation purpose. The computations are then performed to get the required data for Standard Customer Report.

To summarize this section, the approach to extract the required crash information is explained. The existing RSDBnext tool is developed to produce Standard Customer Reports. C# programming language is used to extract and store the data internally.

The result XML file is used to extract the data. This result XML's structure is further explained. This file is parsed to get the required crash data. The next step is to store the data to a suitable file format is discussed further. Then, the importance of using an intermediate step is also discussed.

### *3.2.2* File Format – XML

This sub-section discusses different formats which can be used as the intermediate step. Then, the format for storing crash data is chosen.

The RSDBnext tool is started. Then, all the required data is extracted and stored internally. After this, several computations are performed on this data. This is done to modify the crash data to the required format. Then, this modified crash data is used for generating the Standard Customer Report.

The subsequent step is storing this information in an intermediate file format. The information must then be transformed into the final crash report after being saved in an intermediate file format. There are different formats available for the intermediate step. Amongst the available file formats, the most fitting format for this particular purpose needs to be chosen.

The file formats to be chosen from are XML and JSON. A comparison between the two formats XML and JSON can be seen in Table 3.2.

| **XML** | **JSON** |
|---|---|
| Tags and Attributes handle the data | Data is stored in key-value pairs |
| Example: <name> Max </name> | Example: {“name”:“Max”} |
| Document Oriented | Object Oriented |
| Data is stored in “tree” like data structure | Similar to “map” data structure |
| High security | Less security |
| Only correct data structure is stored | Accepts incorrect data structure |

**Table 3.2: Comparison between XML and JSON Files**

The XML file format stores input data in the form of tags and attributes. The JSON file format stores data in key-value pairs. JSON is less expressive than the XML file format. This means that XML can express more complicated data structures than JSON. For example, the XML file can be stored in a hierarchical format, which is more structured. XML file format is more widely used and supported. This means that there are more tools and libraries for XML file formats. It is also easy to integrate XML with the existing system. [25]

JSON file format also accepts wrong data type as input. In JSON, the input data is not validated. Any incorrect data format can be provided as input. XML provides a set of rules for the input data, which allows for data validation. Data validation is important when the input is crash data. Moreover, XML file format is readable by both humans and machines. XML has built-in namespaces. These namespaces can prevent naming conflicts during runtime. This implies that items with the same name but distinct meanings can be identified. XML file format can be used for easier transformation of data into other file formats.

An example of the XML file format is displayed in Figure 3.6. It can be seen that XML data is more organized and structured. An example of the JSON file format can be seen Figure 3.7. The JSON file format is not structured and this cannot support crash data storage efficiently. These are the drawbacks of using the JSON file format for the crash data.

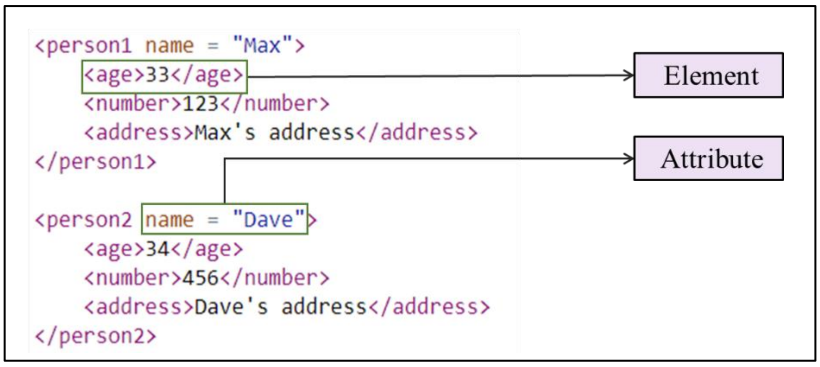
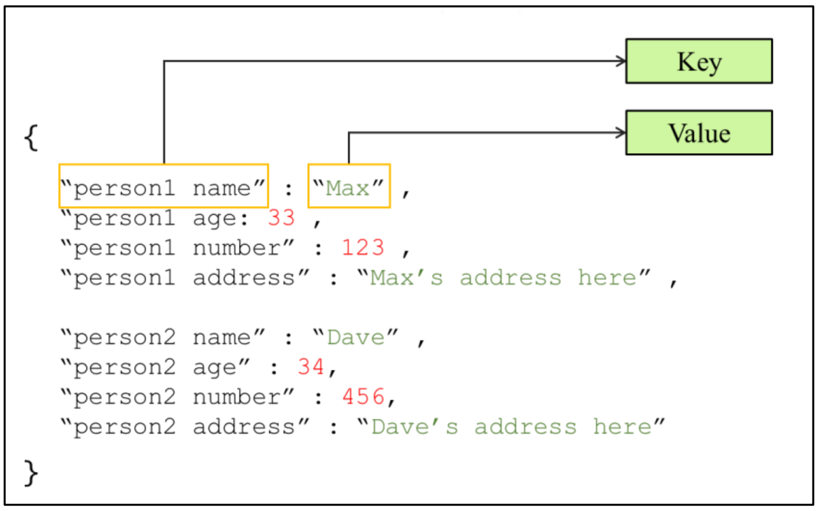

```
<person1 name = "Max">
    <age>33</age>
    <number>123</number>
    <address>Max's address</address>
</person1>

<person2 name = "Dave">
    <age>34</age>
    <number>456</number>
    <address>Dave's address</address>
</person2>
```

**Figure 3.6: Example of an XML File**



```
{
   “person1 name” : “Max” ,
   “person1 age: 33 ,
   “person1 number” : 123 ,
   “person1 address” : “Max’s address here” ,

   “person2 name” : “Dave” ,
   “person2 age” : 34,
   “person2 number” : 456,
   “person2 address” : “Dave’s address here”
}
```

**Figure 3.7: Example of a JSON File**

In Figure 3.6, an example of XML file is depicted. Elements and attributes are used to store data in XML files. Data is represented as a collection of elements in XML. The data is stored in a hierarchical manner in this file format. This hierarchy makes it easy to parse data in XML. In JSON file format, the data is stored as key-value pairs. The flat structure of JSON file format makes it difficult to store complex data. It is possible for JSON file format to represent data using nested objects and arrays. The way data is represented in JSON causes difficulties to parse and manipulate it. JSON lacks a formal schema definition. It means that the structure of data is not clearly defined. This unclear structure in JSON can lead to inconsistencies in data. The inconsistent data can further cause errors in the whole process. Hence, using XML, it is easy to parse and get the required information compared to JSON.

The crash data which is the output of airbag simulation results is hierarchical. To store this data, XML is more suitable. XML file format is better suited as it has better structured format, built-in namespaces, and data validation feature. These features make XML more advantageous than JSON file format. Considering all the advantages and disadvantages, XML file format is more suitable to store the crash data. Therefore, XML file is selected as the file format for this purpose.

The file type in XML needs to be structured effectively. This structure enhances the effectiveness of storing and retrieving data. The structure of the XML file affects the efficiency of the proposed approach. Hence, the structure should be defined correctly. This XML file should then be converted to the required output report.

The structure of the XML file (Standard Customer Report.xml) should be defined. This should be done based on the requirements of the final output. The output Standard Crash Report must be as per the user requirements, with the required crash information. This information includes multiple simulation devices, crashes, variations, and their deployment times. In the final Standard Crash Report, it is required to have different worksheets. The worksheets should be created for different simulation devices. For e.g., one worksheet with all the required crash information should be created for one simulation device. There should also be a summary worksheet. The summary sheet should provide an overview of information from all simulation devices. This means that the summary of all other simulation device worksheets is provided in the summary worksheet. For this reason, it is necessary to structure the XML file according to the requirements of the report.

The input is used to get the necessary crash-related data. This input is an XML file. This input file has a particular structure. This is shown in Figure 3.5. This input XML file is parsed, and the required crash information is extracted. This crash information is stored to an intermediate XML file. Two different structures are defined to store the data to the intermediate XML file. These structures are necessary for the different worksheets required for the final output. The different worksheets are for the number of simulation devices (one worksheet per simulation device) and the summary. Firstly, a structure is created for all the simulation devices. Then, different possible options are checked to store the required values for the summary part in the intermediate XML file.

The proposed hierarchical structure of the Standard Crash Report XML file, for all simulation devices can be seen in Figure 3.8. The defined structure is, for every simulation device, there are multiple crashes. These crashes have many variations with deployment values.

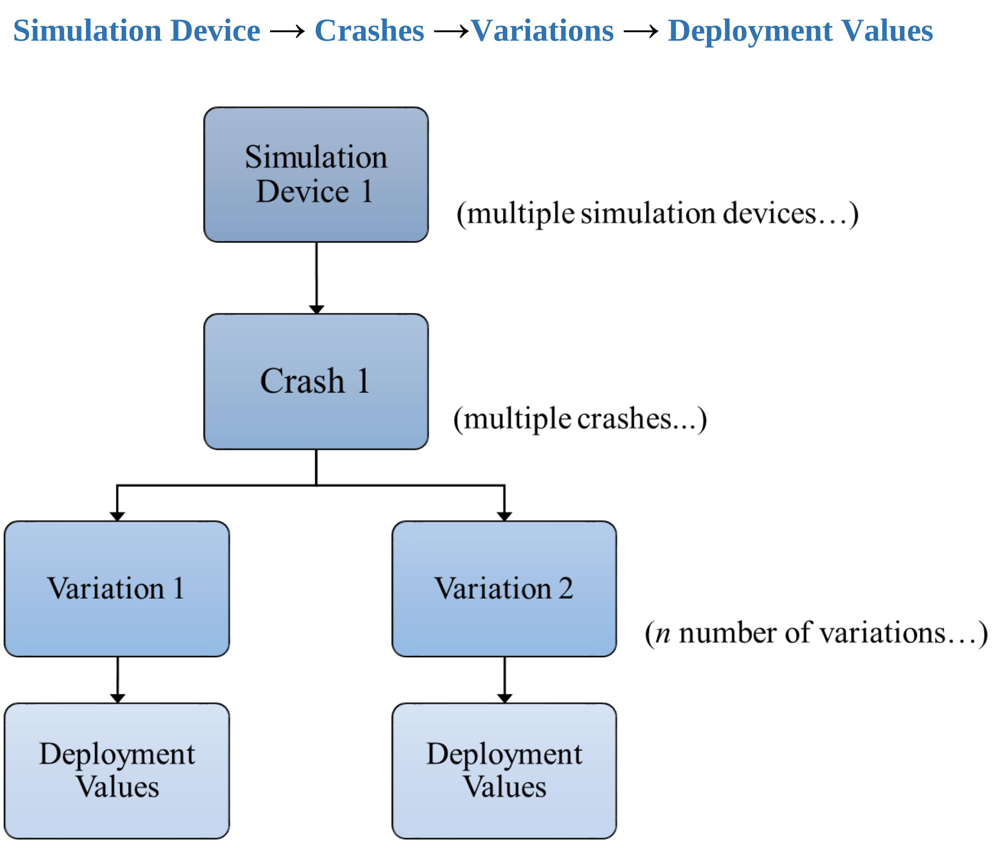

Simulation Device → Crashes →Variations → Deployment Values
Simulation Device 1
(multiple simulation devices…)
Crash 1
(multiple crashes...)
Variation 1
Variation 2
(n number of variations…)
Deployment Values
Deployment Values

**Figure 3.8: Hierarchy of Simulation Devices in the XML File**

The summary sheet must be provided as output to the users along with the simulation device sheets. The XML file should store values for the simulation device sheets and the summary sheet. The required values for the summary sheet needs to be incorporated in the proposed XML structure. There are two ways to do this and these options are discussed in detail further. The two options are as follows:

1) To add the crash information consisting of the summary into the existing XML structure (by adding the data fields with tags along with the already existing tags)

2) To add the crash information with the summary at the end of the existing XML structure (by adding a summary tag at the end of the already existing structure)

The first option is to add only additional required data into the existing structure. For example, each variation contains one airbag fire time. For multiple variations, the average of airbag fire times must be present in the summary sheet. To store the average airbag fire time, another element can be added to the existing structure. This option would complicate the proposed structure of the XML file. In each part, different tags have to be added. This makes the XML structure complex for next steps. It becomes difficult to parse and extract information from this complex XML file. Understanding the XML file also will need a lot of effort and time with this option.

The second possible option is to add a separate summary tag in the end. This would keep the proposed structure undisturbed. In this separate tag, only the required values can be added. The main advantage is that generating Standard Customer Report becomes faster as the XML structure is kept simple. Also, the XML file will still be human readable and understandable. One disadvantage of using the second option is that there will be repetition of two data fields (crash code, simulation device).

After examining the pros and drawbacks of each alternative, the second option is chosen as the better alternative. Repetition of two data fields in the second option is considered. The overall structure is more organized using this approach. The XML file structure is kept simple. It becomes easier to parse this data for the next steps. Retrieving the necessary information for the worksheets becomes simple, though two fields are repeated under the summary tag. Hence, the second option is better than the first option.

The proposed hierarchical structure of the Standard Crash Report XML file, for the summary sheet can be seen in Figure 3.9.

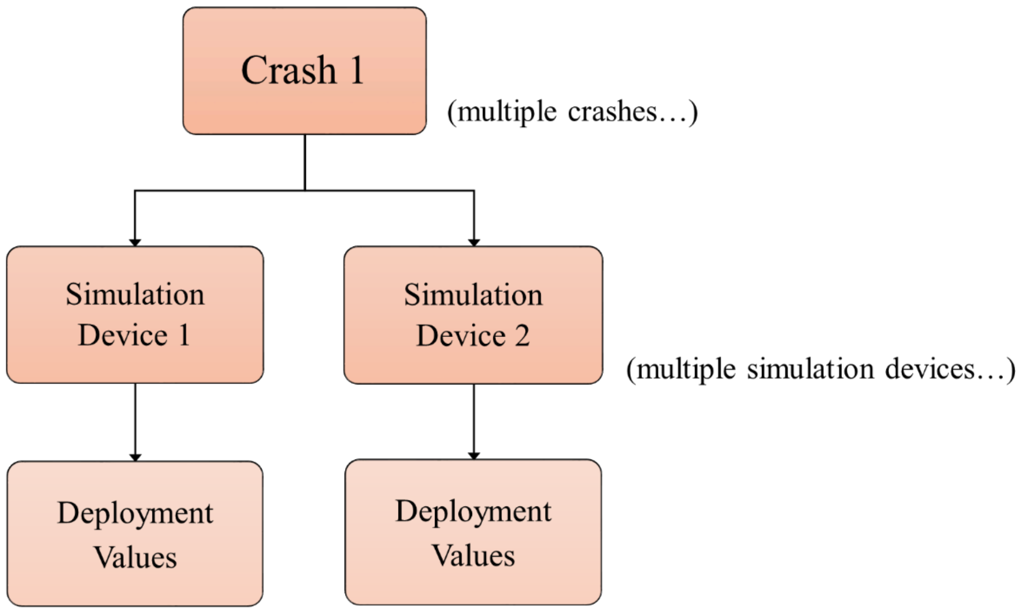


**Figure 3.9: Hierarchy of the Summary Part in the XML File**

This sub-section also explains the importance of using the intermediate step. The programming language used for developing the tool is C#. All the extracted data is stored internally in C#. This internal data is then written out to the XML file. This is performed for various reasons. When the reports are generated directly without the use of an XML file, then the process takes a lot of time. The performance of the system is reduced without using an intermediate step. An intermediate step is required, instead of creating the required Standard Customer Reports directly. Hence, an intermediate step to store the reports is essential. There are disadvantages of generating the reports directly to any formatted output. These drawbacks are as follows:

- **Time Consuming:** The process will become slow and consumes a lot of time if reports are generated directly. Along with it, this process uses a lot of system resources. Also, the response time of the tool increases drastically.

- **Reduced Performance:** Performance is reduced drastically when the size of the data increases. This is because every formatting option and formula must be written individually. This leads to a steep increase in the number of read-write operations. Hence, the overall performance of the tool is reduced.

- **Limited Formatting Options:** The formatting options are reduced when Standard Customer Reports are generated directly. It is difficult to meet user requirements with limited formatting options.

- **Error Prone:** Writing from C# directly can be error-prone. The output documents are often complex. Writing directly to the output report can lead to errors.

- **Version Compatibility Issues:** Writing the output directly can create issues with different versions of the output document. It becomes difficult to ensure the reports are generated as expected.

- **Security Issues:** The system can be exposed to various threats when writing to the output file directly. The system becomes vulnerable to virus and malware attacks.

- **Maintenance:** Writing data directly can make the maintenance process of the tool more difficult. This is because when changes to formatting or structure are required, then these changes have to be done in C# programming. This might be time consuming. It can also cause errors.

Therefore, this intermediate step is necessary. The disadvantages of not having an intermediate step are discussed. To avoid these issues, an intermediate step is crucial for the Standard Customer Report generation process. Therefore, this step helps to enable an efficient report generation process.

In this sub-section, the two possible file formats, XML and JSON are examined. Then, their advantages and disadvantages are presented. After examining both file formats, XML is selected, as it's more suitable for the input crash data and requirements. After selecting the intermediate file as XML, the hierarchy for the crash data is discussed. Then, the importance of having this step is discussed. To conclude this section, the XML file format is required for the generation of Standard Customer Reports.

### *3.2.3* Final Report – Excel

This sub-section discusses the methodology used to generate Standard Customer Reports from the XML file. The possible options for the final output are discussed and an appropriate option is then selected.

The final step is converting the XML file into the required Standard Customer Report. To perform the final step, the output file format must be decided. After that the methodology to transform the XML file to the output file format is discussed. Lastly, the complete approach to generate Standard Customer Reports is presented.

In Section 2.3 of the state - of - the - art chapter, current methodologies to generate reports are discussed. One of the methodology used ARXML as the intermediate file format to store the data. This file is then converted to HTML output. This conversion was performed using JavaScript. Another methodology to generate reports involved using XML as their internal representation. In that methodology, XML-Scripts were used to generate HTML output. Another approach was converting the input text file to output Excel report. After analyzing the current trends of report generation methodologies, HTML and Excel were the two commonly used file formats. The final report was presented to the user using these file formats.

After looking at the most commonly used methodologies, the output file format must be selected appropriately. The output file must be created according to the user requirements, including the required formatting. The user requirements also include the way in which the airbag simulation data is to be presented in the output. For e.g., the first cell in each worksheet should have the name of the simulation device. The output must also include a separate summary section. This summary section should consist of the summary of all airbag simulation devices. After analyzing the user requirements, the most suitable file format needs to be chosen for the output.

The user and customer need to view the simulated crash data. Worksheets are required for the simulation devices and summary as per requirements. This can be accomplished by employing one of the two file formats commonly used to generate reports. The Standard Customer Report can either be generated in Microsoft Excel or HTML. These file formats are discussed further.

Microsoft Excel is a spreadsheet program. This application is well-known for data management and analysis. Microsoft Excel provides a variety of tools for analyzing data. Sorting and filtering the data is possible using Excel spreadsheets. Worksheets can be created in an easy manner. Spreadsheets in Excel may be used to evaluate large amounts of data. This enables the user to find out the correct functionality of the airbag system at different scenarios. Excel has many built-in features which help in formatting the final report. Users can customize the look of the report. It is also easy for the user to format crash data according to requirements. There are numerous options present which are available in Excel for styling purposes. Different fonts and colors can be easily added during the report generation process. Excel can also be easily integrated with other Microsoft Office products. Excel has built-in functions which help in performing computations. This helps users to analyze the crash data.

Web pages are made using HTML files. The developed web pages can be seen on numerous platforms. Using HTML webpages, interactive elements can be added. For example, users can click on certain buttons to display additional content. This can make the webpage more interesting for the user. Third party JavaScript libraries should be used to create worksheets in HTML. Doing this provides some more formatting options. These libraries should be integrated with HTML to provide spreadsheet-like interface on web pages. Some steps are required to do this. The library files should be included in the HTML document. Then, a container element should be defined. Following this, the library should be configured to provide a spread-sheet like appearance. Then, this can be filled with data using the library APIs. Doing this also allows additional formatting options.

This makes the process of creating worksheets using HTML complex. HTML has very less formatting options available. Adding interactive elements using HTML is complex. It has limited flexibility with formatting and layout options. Using HTML as the output file format can lead to issues, like users without permission accessing the page.

After evaluating the benefits and drawbacks of the possible output file options, Excel is selected as the better format. Having worksheets to display the crash data is necessary, and is a requirement. HTML is not capable of creating worksheets in an easy manner, like in Excel. Microsoft Excel has many advantages to have the output in a spreadsheet format. It is also user-friendly. The requirement is also for the users to have the Standard Customer Report in Excel. Hence, Excel is selected as the format for the final Standard Customer Report.

So, the intermediate XML file needs to be converted to Excel. The formatting of the Excel file also needs to be performed. The two approaches to do this are by using XML-Script and XSLT.

XML-Script and XSLT can be used to transform the intermediate XML into the required Standard Crash Report. XML-Script is a scripting language. XML file can be manipulated using XML-Script. This modified XML can be transformed from one file format to the required format. XML-Script is not widely used. It has limited support by other tools and libraries. XML-Script is well suited for flat structured and simple XML data. In addition, utilizing XML-Script to produce reports can make the process more complex and delay its completion. XSLT is a widely used standard language which can be used to transform XML to Excel. XSLT is a powerful language which can be used to generate Standard Crash Reports. XSLT provides better control over formatting options than XML-Script. XSLT also provides support for advanced formatting features. Hence, using XSLT, the transformation is better compared to using XML-Script.

So, the intermediate XML is combined with XSLT to generate the Excel file. Figure 3.10 provides a brief overview of the proposed conversion method. This conversion methodology is further explained in detail.

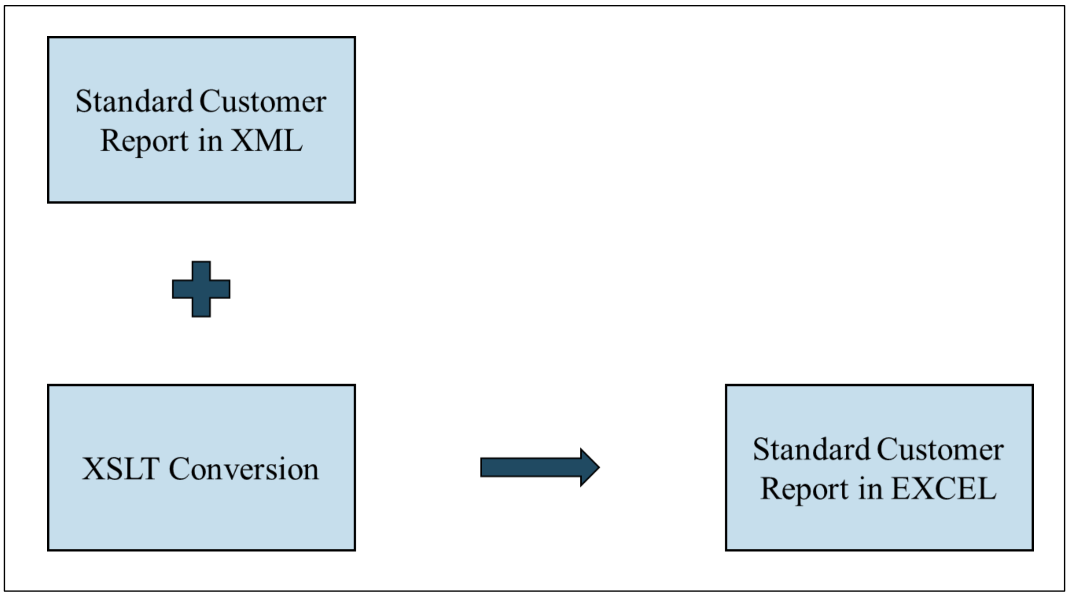

Standard Customer Report in XML
XSLT Conversion
Standard Customer Report in EXCEL

**Figure 3.10: Basic Idea of Conversion to Generate the Standard Customer Report**

The process to convert the XML file to Excel report is performed using XSLT StyleSheet. The required styles are provided to perform this conversion. A collection of instructions are defined regarding the way the XML document should look. These instructions are provided using StyleSheets. StyleSheets can be used to style the XML data by defining the instructions such as formatting and layout. [26]

StyleSheet helps in the formatting and styling of XML elements. It helps in defining the values for each column and row of the Excel file. This StyleSheet needs to be designed as per user requirements. This StyleSheet is combined together with the structured XML data. The StyleSheet and the XML file are converted using XSLT. The StyleSheet needs to be defined for generating summary worksheet and multiple simulation device worksheets. XSLT conversion is performed on the XML file with the defined StyleSheet to generate Standard Customer Reports.

A detailed overview of the proposed Standard Customer Report generation approach is depicted in Figure 3.11. The approach starts from the input database (RSDB). Along with this input, the user provides “Standard” as format input to the RSDBnext airbag application tool to generate the reports.

The tool internally starts running for format option “Result”. This internal running of the tool generates RSDBnext Result Excel file, and RSDBnext Result XML file. The extraction of airbag simulation results is then performed from the RSDBnext Result XML file. The simulated crash data is extracted. This extracted crash data is then stored in an internal structure. Then, computations are performed on the data. This data is stored in an intermediate file, which is of XML format. This Standard Crash Report XML file is combined with a StyleSheet. Then, using XSLT conversion, the required Standard Crash Report is generated. This final output report is an Excel file. This Excel file contains multiple worksheets. The different worksheets are for the simulation devices and summary. This is the proposed concept to generate Standard Customer Reports, which are also known as Standard Crash Reports.

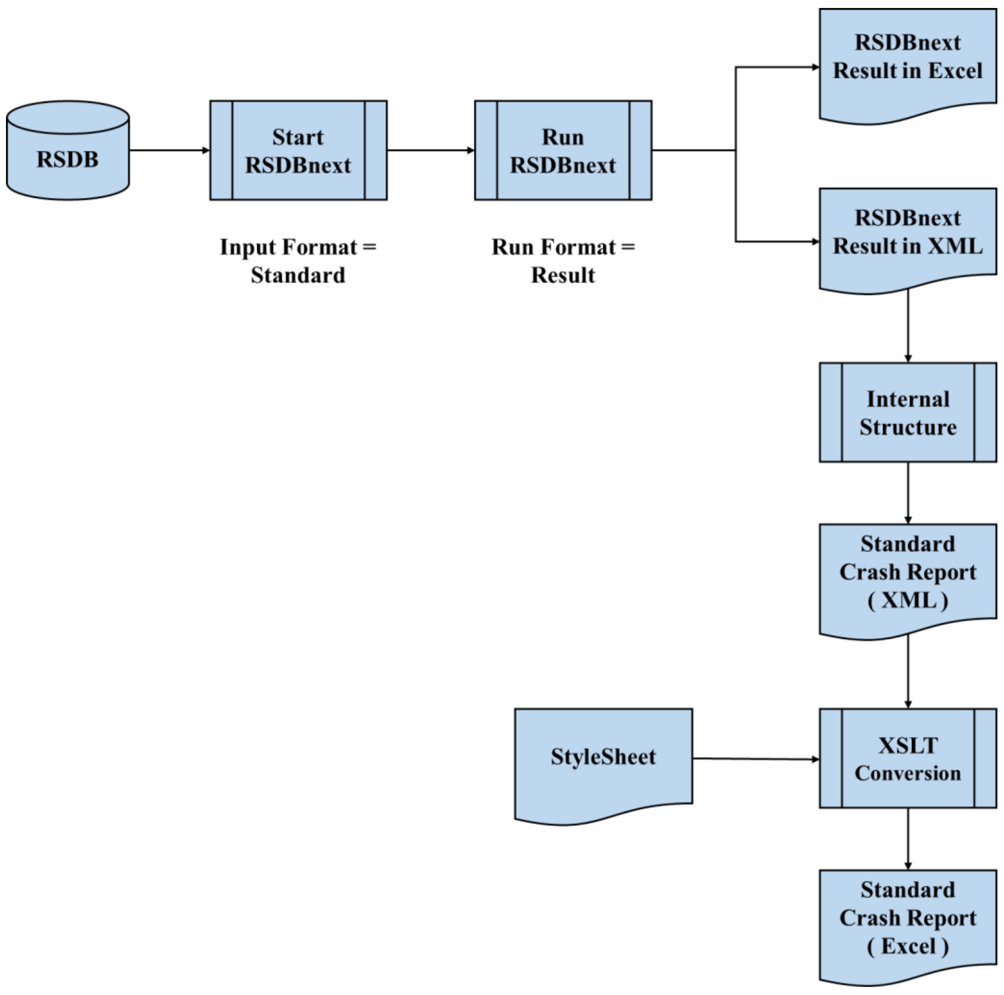


**Figure 3.11: Concept to Generate Standard Customer Reports**

## 3.3 Summary

In this chapter, the existing methodology to generate Standard Customer Reports is discussed in Section 3.1. Then, the existing methodology is also reviewed for its limitations. A novel approach for generating the required reports is proposed. This approach overcomes the limitations of using Excel macros to generate reports.

In Section 3.2, a concept is proposed for generating Standard Customer Reports using airbag simulation results. In Section 3.2.1, the possible ways to handle airbag simulation results are discussed, and an appropriate method is selected. In the next section (Section 3.2.2), the need for an intermediate file is demonstrated. The intermediate file format is chosen as XML. This is mainly due to its hierarchical and structured data storing method. Then, the XML structures for storing crash data for simulation device worksheet and summary worksheet is presented. This hierarchy is required to store the simulated crash data. In Section 3.2.3, the final report generation process is discussed. The suitable file format is chosen, which is Excel. This Excel file contains different worksheets. These are the simulation device worksheets, and a summary worksheet. Finally, a detailed overview of the entire report generation is depicted in this section. Overall, the proposed concept provides a methodology to generate Standard Customer Reports in an efficient manner. The approach is also designed for overcoming the shortcomings of the current method. The implementation of the proposed approach to generate the required reports is covered in the next chapter.

# 4 Implementation

The implementation of the developed concept for generating Standard Customer Reports is detailed in this chapter. In the previous chapter, a methodology was proposed for generating these reports. The required tools and file formats used for the approach are discussed. In the implementation chapter, the explained tools and the selected file formats are used to generate the required Standard Customer Reports.

The chapter on implementation is split into three major components. This chapter's first part covers the airbag tool's input. The way to start the airbag tool, and extract data from the input is discussed. Then, in second part the computations that were done on the extracted data are discussed. This portion is subdivided into two parts. The first sub-section discusses about the check performed for having equal amplitude steps. In the next sub-section, the fire behavior of the airbag is explored. In the third section, the implementation of output files from the airbag tool is discussed. Also, the simulation device worksheets and summary worksheet are examined. An overview is given in the chapter's final section.

## 4.1 Input to Steer the Airbag Tool

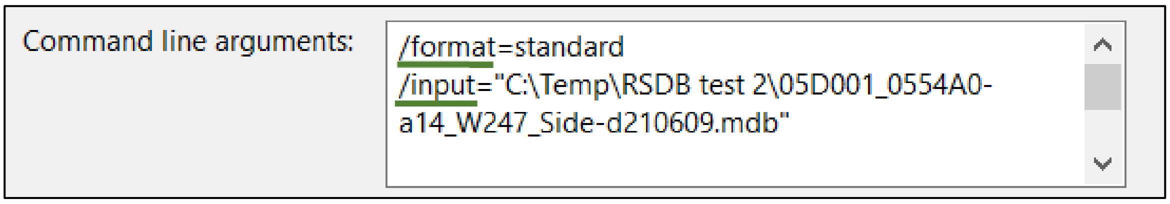
This section discusses the inputs provided to the airbag application tool to generate Standard Customer Reports. Using these inputs, the data extraction ways are also explored.

The airbag application tool named RSDBnext is a data extraction tool. From the database, RSDB, data is extracted using this tool. This tool is built using C# programming language in Visual Studio IDE. There are multiple inputs which can be provided to the tool. The first input to the RSDBnext tool would be the database file location. This database is stored as .mdb file, which is a Microsoft Access file. This database file path is provided as one of the inputs to RSDBnext tool. Along with the database file path, RSDBnext also requires the format option to be specified. There are four possible format options to start the RSDBnext tool. These format options are discussed in detail in the Section 3.2.1. Along with the .mdb input path and the format option, the RSDBnext tool also requires few other inputs such as output file path and update option. The inputs to the RSDBnext tool are provided as command line arguments. These input options can be seen in Table 4.1.

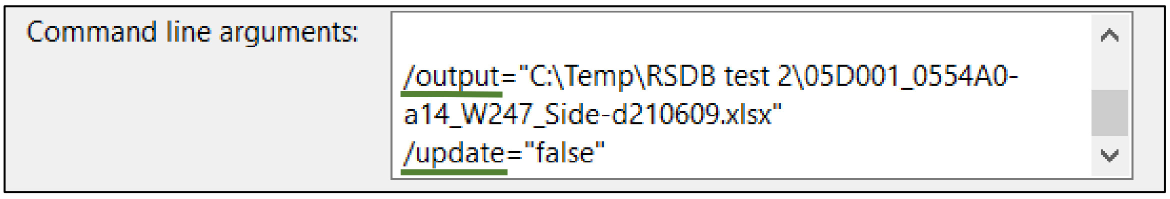
| Command Line Arguments | Description |
|---|---|
| /format | Input provided to the RSDBnext tool to determine the type of the output report. |
| /input | The database (.mdb) file path is the input file |
| /output | The Excel file path for the output |
| /update | This field can either have true or false. The latest version of the tool is installed based on this field. |

**Table 4.1: Inputs to the Airbag Tool**

An example of the command line arguments provided to start the RSDBnext tool can be seen in Figure 4.1.


Command line arguments:

```
/format=standard
/input="C:\Temp\RSDB test 2\05D001_0554A0-
a14_W247_Side-d210609.mdb"
```


Command line arguments:

```

/output="C:\Temp\RSDB test 2\05D001_0554A0-
a14_W247_Side-d210609.xlsx"
/update="false"
```

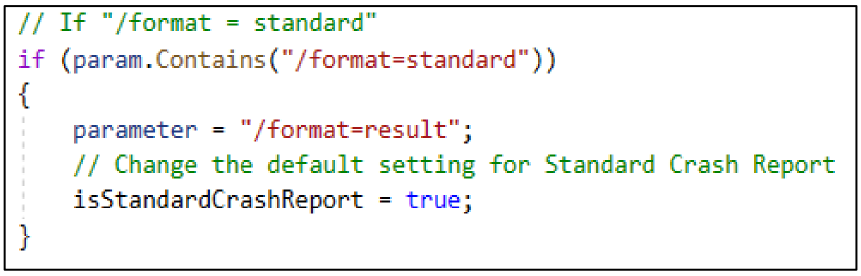
**Figure 4.1 Command Line Arguments to Start the RSDBnext Tool in Visual Studio**

These options in the command line arguments are:

- The first input required is the format option.
- Then, the input file path is checked if it contains a database at the given location.
- The output path is required by the RSDBnext tool to write out the files at that particular location. This output file path is of an Excel file path with the extension as .xlsx.
- Then, the last input required to start the RSDBnext tool is the update value. This is either true or false. The latest version of the RSDBnext tool is installed if the update value is set to true. After updating the tool, further process is continued with the latest version.

These are the input fields which need to be provided by the user to start the tool. To generate Standard Customer Reports, the user needs to provide input format option as “standard”. The input is checked in C# programming. The code snippet to perform this action is illustrated in Figure 4.2. If the input contains the format option as “standard”, then it will set the format option to “result”. Additionally, a flag is set to differentiate between “result” and “standard”. Once the format option is set to “result”, the RSDBnext tool runs with this format option and generates the result XML and result Excel file as output. This step is performed according to the proposed concept.

The generated result XML file contains all the required crash data to generate Standard Customer Reports. The required crash data needs to be extracted from the result XML file. This step is performed without disturbing the functionality of the RSDBnext tool. It means that other format options such as scaling and misuse must work like before.



```
// If "/format = standard"
if (param.Contains("/format=standard"))
{
    parameter = "/format=result";
    // Change the default setting for Standard Crash Report
    isStandardCrashReport = true;
}
```

**Figure 4.2: Code Snippet to Start RSDBnext with “/format = standard”**

Figure 4.3 illustrates the process of starting the RSDBnext tool. The format option is checked as a first step. If the format is “result”, then result XML is generated. After the result XML file is generated, the format option is set back to “standard”. The result of RSDBnext, which is the XML file is used to generate the Standard Customer Reports. The user does not have to do any additional steps to provide this information.

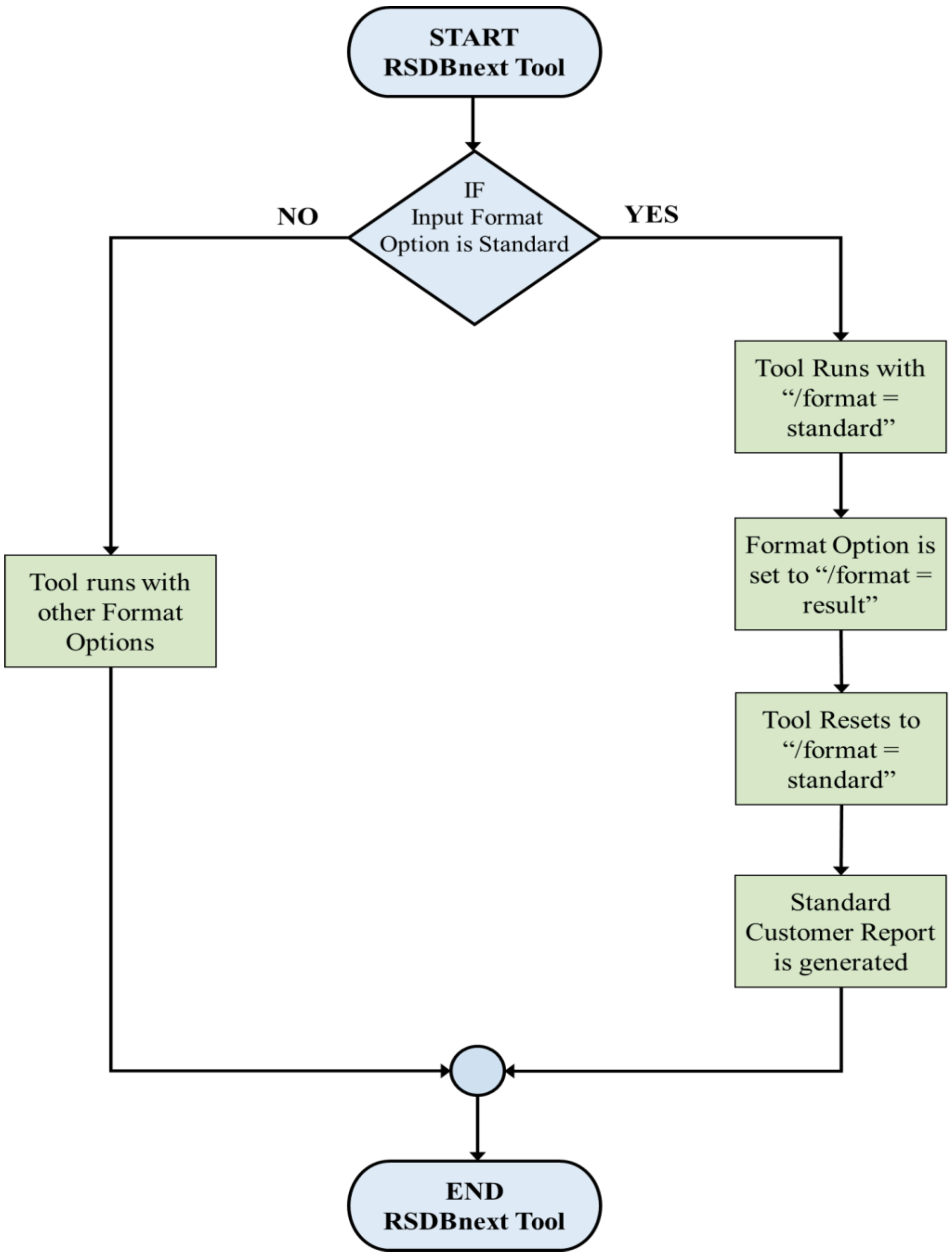
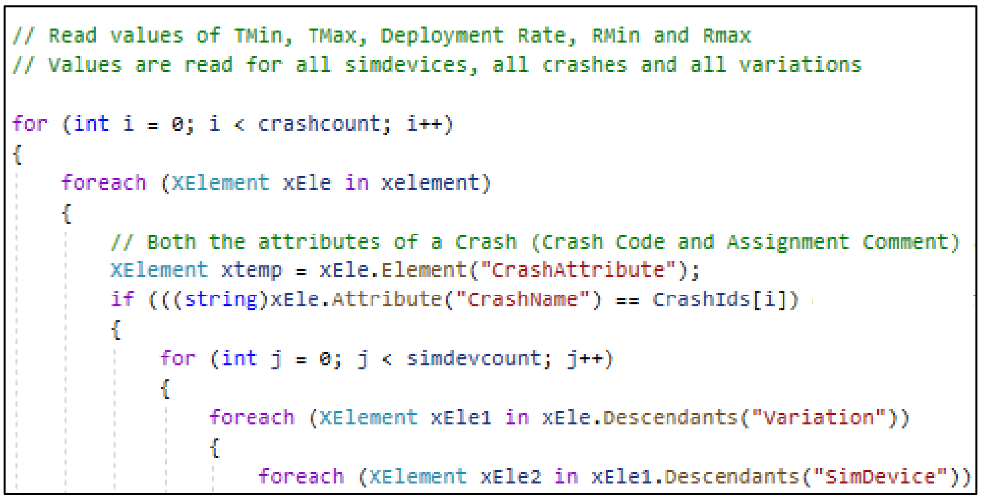


**Figure 4.3: Flowchart to Start RSDBnext Tool**

After the result XML file is generated, the next step is to extract the required crash data from this file. The extraction of crash data from the result XML file is done using C# programming. Xelement is an XML parsing API (Application Programming Interface) which is used in C# to parse the XML file. Xelement can be used by adding System.Xml.Linq header to the C# class. Xelement is used because parsing the hierarchical data in result XML file is easy. [27]

The result XML file contains crash data in a hierarchical format. This hierarchical format was shown in Figure 3.5 (Chapter 3). The structure of the data is: for each crash, there are multiple variations. For each variation, there are many simulation devices. Then, for each simulation device, deployment values are stored. The deployment values comprises of the Tmin, Tmax, Rmin, Rmax, and deployment rate values. The code snippet depicted in Figure 4.4 shows the way to parse the XML document. The crash data is stored in the form of string in XML format. The values of Tmin, Tmax, Rmin, Rmax, and deployment rate values are stored in string form. So, it must be converted to numerical values without any errors. Any error in these steps can lead to incorrect Standard Customer Reports. The airbag simulation results need to be extracted correctly. Hence, "CultureInfo.InvariantCulture.Numberformat" is used to make sure that the extracted values are correct. "CultureInfo.InvariantCulture.Numberformat" is a feature of C# programming. Using this feature ensures that the numbers are formatted accordingly and correct results are produced [28]. For e.g., the decimal point for a German system is set to ',' , whereas for other countries it is set to '.'.



```
// Read values of TMin, TMax, Deployment Rate, RMin and Rmax
// Values are read for all simdevices, all crashes and all variations

for (int i = 0; i < crashcount; i++)
{
    foreach (XElement xEle in xelement)
    {
        // Both the attributes of a Crash (Crash Code and Assignment Comment)
        XElement xtemp = xEle.Element("CrashAttribute");
        if (((string)xEle.Attribute("CrashName") == CrashIds[i])
        {
            for (int j = 0; j < simdevcount; j++)
            {
                foreach (XElement xEle1 in xEle.Descendants("Variation"))
                {
                    foreach (XElement xEle2 in xEle1.Descendants("SimDevice"))
```

**Figure 4.4: Extract Crash Data from the Result XML**

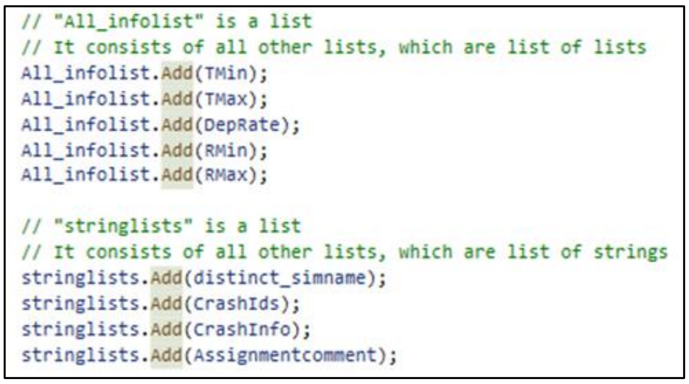
The extracted values are then converted to numerical values in the correct format. These values then need to be stored in an internal data structure. Along with the numerical values of deployment rates, additional data also needs to be extracted to generate the required reports. The additional data includes crash id, simulation device name and many other fields. The numerical values and the additional data need to be stored in the internal memory. This internal storage is performed for an overall efficient process. If the file is read and written directly to the intermediate file, then this could cause issues, such as a complex report generation process. Therefore, it is necessary to store the data in the internal memory for efficient processing.

To store the extracted results, an internal data structure is selected. List<double> [] [] is a two dimensional array of a list of doubles. This is used as the internal structure for numerical values, such as Tmin, Tmax, Rmin, Rmax, and deployment rate. Double is used as the data type because the decimal numbers of the values should be stored. A list structure is considered because it is a powerful data structure. The lists are also dynamic in size. So, the two dimensional structure is considered because of its flexible data organization. This structure was chosen because of its flexibility to organize the extracted crash data.

Furthermore, a list of strings data structure is considered for string values. To store all the required crash data, this data structure is named accordingly. For e.g., all the extracted Tmin numerical values are stored in a list of integers structure. This list is named as “Tmin”. Likewise, the extracted string values such as crash information are stored as “CrashInfo”. The appropriate naming of the data structure is important. This helps to have a clean code which can be maintained, and developed further easily.

To further optimize the internal data structure, list of list (double values) and list of list (strings) is developed and used. This list of list structure contains all the two dimensional double lists such as Tmin, Tmax, Rmin, Rmax and deployment rate. This means that a larger list stores the developed list structure. Figure 4.5 depicts the list of list (integers) and list of list (strings).

First, the XML file is read and all the values are added to the respective numeric list. The next step is to add all the numeric list structures to a larger list. The same process is also performed for all the string lists. This larger list is required for easier data exchanges from one class to the other class in C#. It is important to maintain the order of lists which are stored in the larger structure for further implementation steps.



```
// "All_infolist" is a list
// It consists of all other lists, which are list of lists
All_infolist.Add(TMin);
All_infolist.Add(TMax);
All_infolist.Add(DepRate);
All_infolist.Add(RMin);
All_infolist.Add(RMax);

// "stringlists" is a list
// It consists of all other lists, which are list of strings
stringlists.Add(distinct_simname);
stringlists.Add(CrashIds);
stringlists.Add(CrashInfo);
stringlists.Add(Assignmentcomment);
```

**Figure 4.5: Internal Data Structure**

The extraction of crash data from the input is one of the important steps in the process of report generation. The crash data is extracted and then, converted to the required format. After transforming the data into the necessary format, it is saved to the data structure. This data structure is important for performing computations on the simulated crash data. In the following section, the computations performed on simulated crash data are discussed.

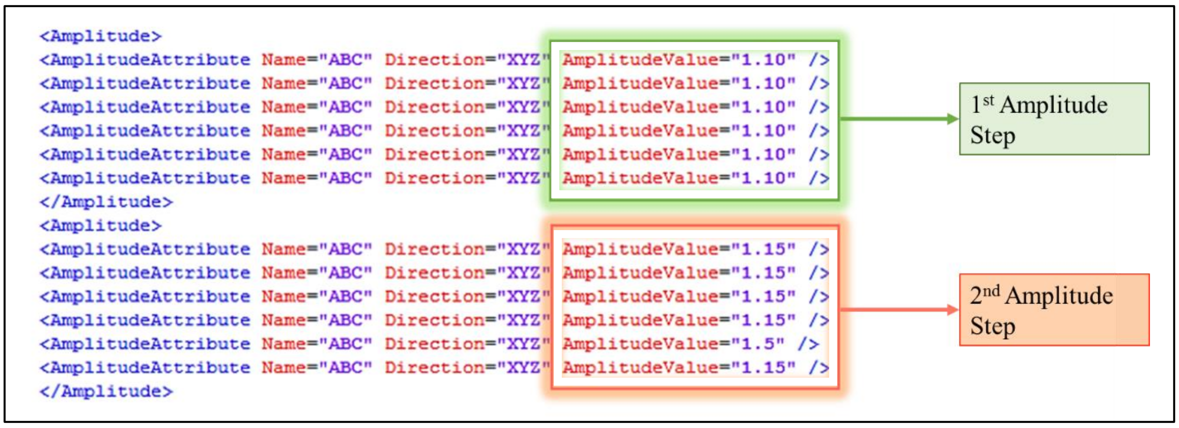
## 4.2 Performing Computations on the Data

The computations performed on the data before generating Standard Customer Reports is discussed. The first sub-section discusses the checks performed for equal amplitude steps. A method to handle the case when amplitude steps are not equal is developed. In the next sub-section, the fire behavior of the airbag is calculated based on the input data. Along with this, certain computations which are to be performed for the worksheets in the Excel file are discussed.

The required crash data which is extracted is stored in the defined internal data structure. This extracted data needs to be adapted to be presented for the Standard Customer Reports.

### *4.2.1* Check for Equal Amplitude Steps

The RSDBnext is started with the format option as “standard”. The tool internally starts running for format option “result” and generates result XML file. This result XML file which is the input XML file, contains multiple amplitude steps. Each step of the amplitude contains multiple amplitude values. These amplitude values in each step must be equal. An example of amplitude steps in the input XML file can be seen in Figure 4.6.



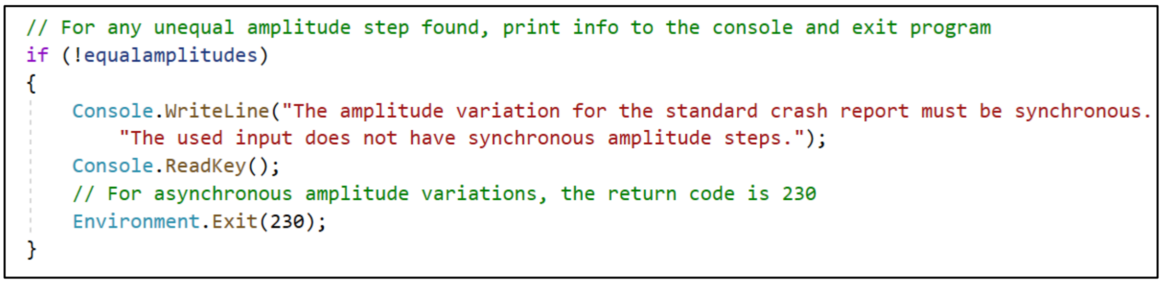
```
<Amplitude>
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.10" />
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.10" />
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.10" />
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.10" />
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.10" />
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.10" />
</Amplitude>
<Amplitude>
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.15" />
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.15" />
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.15" />
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.15" />
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.5" />
<AmplitudeAttribute Name="ABC" Direction="XYZ" AmplitudeValue="1.15" />
</Amplitude>
```

**Figure 4.6: Amplitude Steps in the Input XML file**

The first amplitude step in the figure has multiple amplitude values. These amplitude values are all equal. The second amplitude step contains five equal values and one different value. This amplitude step has unequal values. Even if one amplitude step has unequal values, then it is considered as an unequal amplitude variation. Hence, the example in Figure 4.6, has unequal amplitude variation because of one value being 1.5 and the others 1.15 in the second amplitude step.

To generate Standard Customer Reports, the amplitude steps should contain all equal amplitude values. A check needs to be performed before generating the required report. This check needs to be performed, and then the reports should be generated if the input amplitude steps are synchronous. If the amplitude variations are asynchronous, then this case should be handled by the RSDBnext tool.

To perform this check, a method is created in C#. This method parses the input XML file, where the amplitude steps are present. Then, all the amplitude values of each amplitude step are stored in a list structure. After the values are stored, a check is performed on this internal structure for unequal amplitude values. If the program did not find any unequal amplitude values, then the Standard Customer Report will be generated. If the program finds any  unequal amplitude values, then the program will stop running. The reason for terminating the application will be stated in a message written on the console. Along with the message, an exit code is printed on the console. The message on the console and the exit code will help the user resolve the issue. Figure 4.7 depicts the code snippet to perform the check for equal amplitudes. A message is also displayed to the user, along with the exit code.



```
// For any unequal amplitude step found, print info to the console and exit program
if (!equalamplitudes)
{
    Console.WriteLine("The amplitude variation for the standard crash report must be synchronous.
        "The used input does not have synchronous amplitude steps.");
    Console.ReadKey();
    // For asynchronous amplitude variations, the return code is 230
    Environment.Exit(230);
}
```

**Figure 4.7: Code Snippet to Print an Error Message to the Console**

For displaying the message on the console, “Console.WriteLine” is used. By displaying this message, the user can be notified that the input does not have synchronous amplitude steps. The exit code, also known as return code is also displayed. This return code can provide more information about the status of a tool. A return code can indicate whether the program has completed successfully or if there were any errors or exceptions. Table 4.2 depicts the error codes or exit codes which are used in the RSDBnext tool.

| Exit Codes | Description |
|---|---|
| 0 | RSDBnext worked like expected |
| 50 | RSDBnext needs to be updated |
| 200 | Output path does not exist |
| 230 | Amplitude variation must be synchronous |
| 255 | Fatal error |

**Table 4.2: RSDBnext Exit Codes**

First, the tool checks for equal amplitude steps. If the amplitude values are synchronous, then the tool generates the Standard Customer Report. As per the user requirements, the ampitude values should be present in the Standard Crash Report. So, along with the check of ampitude steps, the amplitude values are extracted for the report generation. After they are extracted, the values are converted to percentages.

For example, in Figure 4.5 the amplitude values for the first amplitude step is “1.10”, this should be stored as “110%” internally. This conversion of value to percentage is performed for all the amplitude steps. All these values are stored in lists and are used for producing the required report. The code part to perform the step of converting the amplitude values to percentage values can be seen in Figure 4.8.

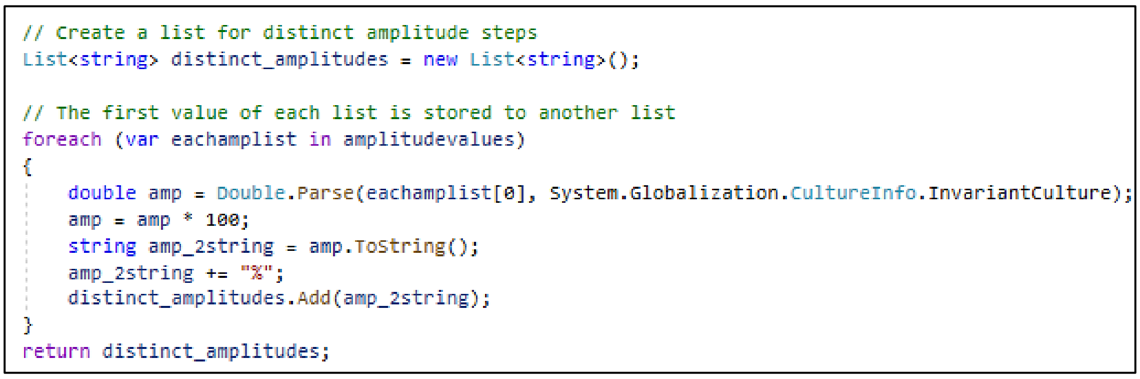

```
// Create a list for distinct amplitude steps
List<string> distinct_amplitudes = new List<string>();

// The first value of each list is stored to another list
foreach (var eachamplist in amplitudevalues)
{
    double amp = Double.Parse(eachamplist[0], System.Globalization.CultureInfo.InvariantCulture);
    amp = amp * 100;
    string amp_2string = amp.ToString();
    amp_2string += "%";
    distinct_amplitudes.Add(amp_2string);
}
return distinct_amplitudes;
```

**Figure 4.8: Snapshot for Synchronous Amplitudes**

The check for equal amplitude values is performed. If the amplitudes are synchronous, then the values are converted to the required format. Then, the next step would be to perform computations on Tmin, Tmax and deployment rate values. This will determine the fire behavior. The fire behavior will be discussed in the coming sub-section.

### *4.2.2* Fire Behavior

In the output Excel file, each of the simulation device worksheet requires multiple columns, named under “ALL”. This “ALL” section contains “YES” or “NO” filled for each amplitude step. If “YES” is present in the cell, it represents that for the particular amplitude step, there was an airbag deployment for all time variations, for that particular crash. Likewise, “NO” represents that there was no airbag deployment for that simulation. The values for these columns are to be computed from the extracted crash data.

To compute “ALL” values for this section, the deployment values which are extracted is used. The deployment rate contains percentage of airbag deployment for that particular scenario. Based on these percentage values, the values for “ALL” section is decided. The deployment rate values and their depiction can be seen in Table 4.3.

| Deployment Rate | Fire Behavior | ALL Value |
|---|---|---|
| 0 | No Fire | No |
| >0 and <100 | Partial Deployment | No |
| 100 | Full Fire | Yes |

**Table 4.3: Determining the Fire Behavior and ALL Values**

Based on the deployment rate value, the “ALL” values and fire behavior of that scenario is decided. Whenever the deployment rate is not equal to 100, then the “ALL” value is a “no”. This needs to be displayed in the output Excel file. The user and the customer can see for a particular amplitude step, if the airbag is being deployed for all time variations or not.

Along with the “ALL” values, a specific coloring is required for certain scenarios. The scenarios where coloring is required are as follows:

1. The “ALL” value is “NO”
2. The Tmin value or Tmax value should not be equal to zero

When both the conditions (1 and 2) are satisfied, then yellow color must be applied. This needs to be applied in the cell, for that particular amplitude step. The fields where yellow should be applied are below the Tmin, Tmax, Rmin, Rmax and ALL values in the Excel file.

Adding colors by checking the specific condition can be performed in one of two possible ways. The two options through which adding colors is possible are:

1) To perform the condition checks in StyleSheet, and apply yellow color using StyleSheet itself
2) To perform condition checks in C#, and apply yellow color using StyleSheet

These are the two possible ways which can be used to apply the required formatting to the output file. The first option is to perform the checking of conditions, and apply the required formatting using StyleSheet only. The second option is to perform the checks in C#, then adding an attribute to that particular element. By adding an attribute, the StyleSheet can identify if yellow color needs to be added. So, checks are performed in C#, then yellow color style is applied using StyleSheet. The first option has a few drawbacks. They are as follows:

- **Complex StyleSheet:** Performing condition checks in StyleSheet can make it complex and hard to maintain. When the StyleSheet becomes complex, it can affect the performance of the tool.

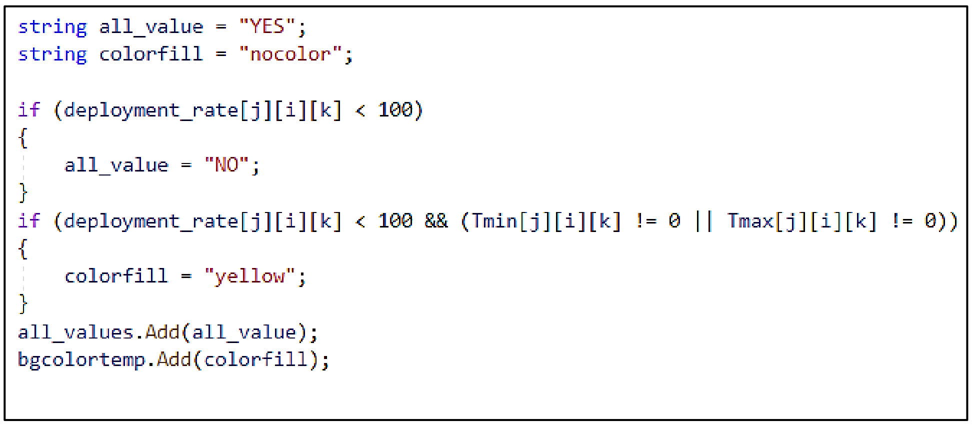
- **Limited Control over Condition Checks:** If the condition checks to be performed are complex, such as requires more computations, it becomes difficult to compute these condition checks in the StyleSheet.

The advantage of the second option is:

- **Code Organization:** Separating the condition checks from the formatting part makes the code more organized and easier to maintain. By performing the checks in C#, it is easier to update or modify the code without affecting other parts in the code.

The second option makes the computations faster, as the checks are performed in C#. By choosing the second option, and keeping the StyleSheet less complex, the code can be easily maintained. Hence, the second option is selected to add yellow color to the required cells.

In Figure 4.9, the code snippet depicts the checks done to determine the “ALL” values, and also the checks to apply yellow color. The necessary computations are performed. Then, the results are stored in lists. For e.g., A list named “bgcolortemp” contains either “nocolor” or “yellow”. This list contains either one of the values for all the simulated crashes. These values are added to the XML file as attributes to the elements. The StyleSheet looks at these attributes, and decides whether yellow color should be added or not.



```
string all_value = "YES";
string colorfill = "nocolor";

if (deployment_rate[j][i][k] < 100)
{
    all_value = "NO";
}
if (deployment_rate[j][i][k] < 100 && (Tmin[j][i][k] != 0 || Tmax[j][i][k] != 0))
{
    colorfill = "yellow";
}
all_values.Add(all_value);
bgcolortemp.Add(colorfill);
```

**Figure 4.9: Code Snippet to Compute ALL Values and Check for Yellow Color**

The “ALL” values are computed and stored in a list structure. These values are computed for simulation device worksheets. Along with “ALL” values, the fire behavior values were also computed. The fire behavior values (Table 4.3) are required for the summary worksheet.

The logic for marking yellow color is different for simulation device worksheets and for summary worksheet. The requirement for yellow color in simulation device worksheet is that the deployment rate is not 100. The requirement for marking yellow in summary worksheet is discussed further.

The requirement is that the summary worksheet must contain columns for “Max”, “Tmax Nominal”, “Min” and “Fire Behavior” for each simulation device. Values are computed for each crash, and for each simulation device from the internal data structure. They are calculated in the following manner:

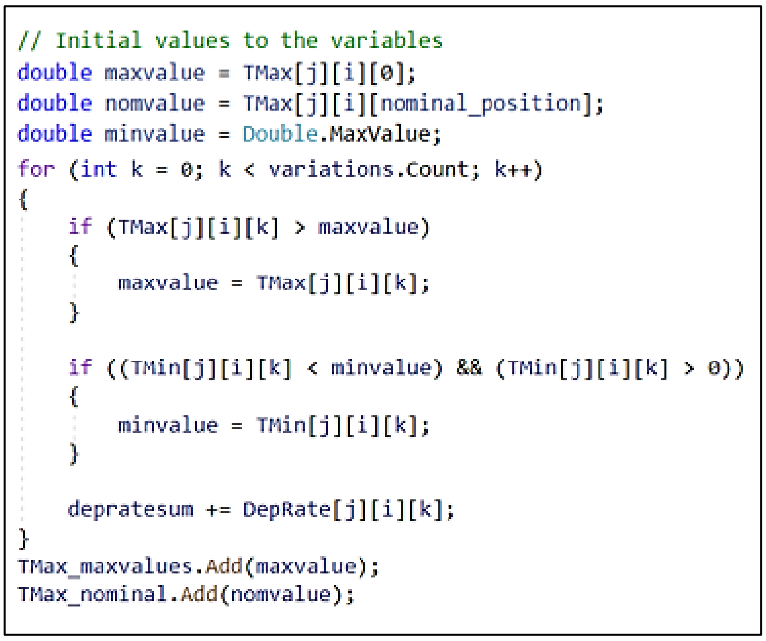
- **Max:** Max values are obtained by computing the maximum of Tmax values for all variations. For e.g., There are 10 variations, and each variation has a Tmax value. The Max value for the summary sheet is computed by performing the maximum of Tmax values for the 10 variations.

- **Tmax Nominal:** The nominal position is when the amplitude step is at 100%. Tmax nominal is found out by taking the value of Tmax at the nominal position. The values to this field are computed as “nomvalue”, and stored in a list structure named as “Tmax_nominal”.

- **Min:** When compared to the Max values, the Min values for the summary sheet are computed differently. The Min values is the minumum Tmin value of all the variations, excluding 0. If there is no airbag fired for that particular variation, then the value will be 0. This value should not be included for computing the Min value. To perform this computation, the largest value which is possible for the double data type is taken by using “double.MaxValue”. This largest value is intialized to the Min value at the begininng. Then the internal data structure is checked for a smaller value than the initialized one. This check is performed for values greater than zero. Then, the Min value is updated with the smallest value.

- **Fire Behavior:** The fire behavior field is determined based on the deployment rate of all the variations. The average of all the values of deployment rate is calculated. This value is checked if it's 0, 100 or any other number between 0 to 100. Then, based on this computation, the fire behavior is determined.

These are the necessary computations which have to be performed on the extracted crash data. They are done to obtain the required values for the summary worksheet. The computations performed can be seen in Figure 4.10.



```
// Initial values to the variables
double maxvalue = TMax[j][i][0];
double nomvalue = TMax[j][i][nominal_position];
double minvalue = Double.MaxValue;
for (int k = 0; k < variations.Count; k++)
{
    if (TMax[j][i][k] > maxvalue)
    {
        maxvalue = TMax[j][i][k];
    }

    if ((TMin[j][i][k] < minvalue) && (TMin[j][i][k] > 0))
    {
        minvalue = TMin[j][i][k];
    }

    depratesum += DepRate[j][i][k];
}
TMax_maxvalues.Add(maxvalue);
TMax_nominal.Add(nomvalue);
```

**Figure 4.10: Computations for Summary Worksheet**

After performing computations for the summary sheet, additional formatting should also be added. Additional formatting such as yellow background color must be added to the summary worksheet. The yellow should be marked as the background color of the cell based on the following conditions:

- **Min, Max and Fire Behavior Fields:** If the deployment rate of all variations is neither 0 nor 100, then these fields must be marked with yellow background. In other words, when the fire behavior is “Partial Deployment”, then the cell needs to have a background color.

- **Tmax Nominal field:** Tmax nominal is the value at an amplitude step of 100%. If the deployment rate for this particular variation is not equal to 100 or 0, then this field should be marked with yellow background color. The formatting option for this field is computed separately.

The computations on the internal data structure are performed. The results are then stored in a different structure internally. The results of the computations are used as attributes. They are appended to the XML file's components. These attributes are used to format the worksheets. Based on the attributes, whether to add yellow background color or not is decided by the StyleSheet.

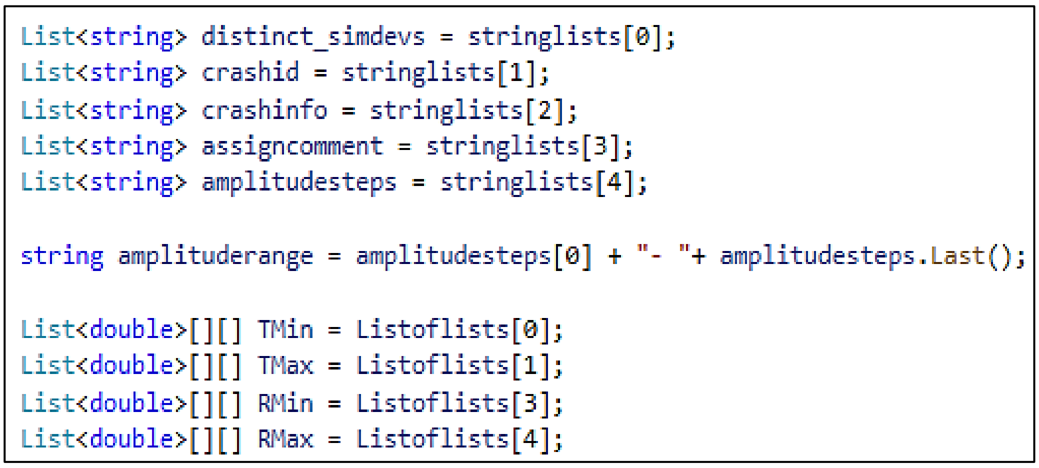
These are the computations that must be done on the internal data structure. The retrieved data is modified in accordance with the worksheets' specifications. Then, the next step is to generate the XML file, and the required Standard Customer Report from this computed data. The implementation of the next steps is covered in the following part.

## 4.3 Implementation to Generate Output Files

The necessary computations are performed on the internal crash data. This data is ready to be transformed to the XML file. Then, this XML file has to be converted to simulation device worksheets, and the summary worksheet in the Excel file. The conversion step is important for generating Standard Customer Reports in a fast and efficient manner.

First, the data from the internal structure needs to be sent to a different class in C#. There are different classes in C#. The separation of having different classes is done to ensure that the developed code is easy to understand and maintain. [29]

The internal computed data is sent to the class named “Standard_XmlWriter.cs”. This class is created to write the data to the XML file. The computed data such as list of lists structure and list of strings structure is sent to this class. The list of strings data structure contains individual lists of each of the simulation devices, crash information and so on. It is necessary for the lists to be extracted in the same order like they were added before. The order in which the lists were added to the structure was shown earlier (Figure 4.5). In Figure 4.11, the order in which the lists are extracted for usage can be seen. The order of lists is the same in both the figures.



```
List<string> distinct_simdevs = stringlists[0];
List<string> crashid = stringlists[1];
List<string> crashinfo = stringlists[2];
List<string> assigncomment = stringlists[3];
List<string> amplitudesteps = stringlists[4];

string amplituderange = amplitudesteps[0] + "- "+ amplitudesteps.Last();

List<double>[][] TMin = Listoflists[0];
List<double>[][] TMax = Listoflists[1];
List<double>[][] RMin = Listoflists[3];
List<double>[][] RMax = Listoflists[4];
```

**Figure 4.11: Extracting Lists from the Internal Data Structure**

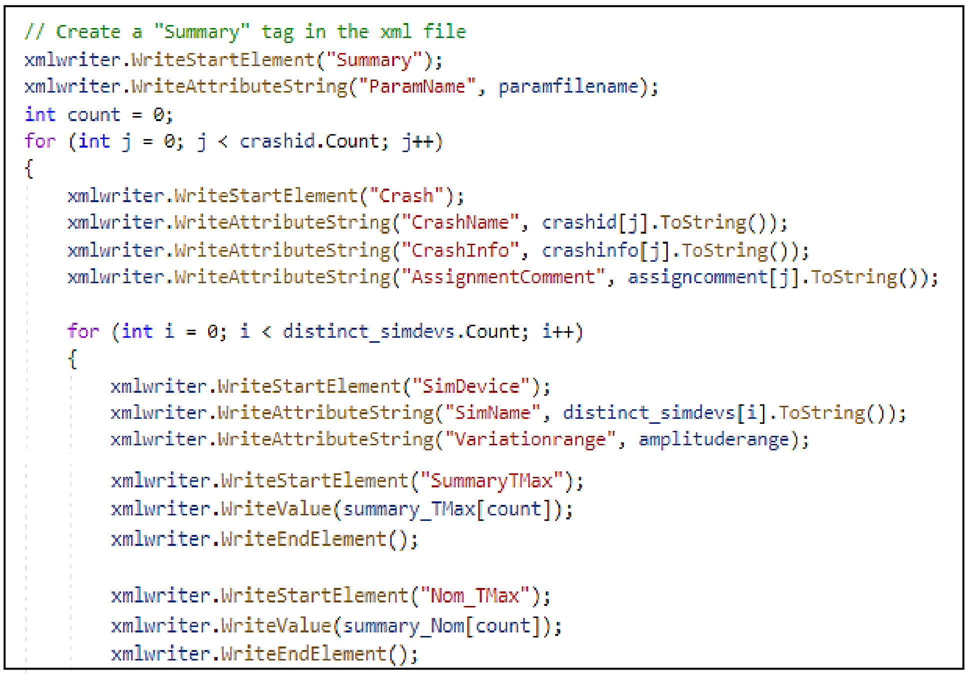
The required data is extracted in the “Standard_XmlWriter.cs” class. Then, this data is written to an XML file. Along with the creation of the required XML file, the StyleSheet is also discussed further.

The XML file created, contains two parts. The first part contains multiple sub-parts. Each sub-part contains data for one simulation device. The second part of the XML file consists of data for summary worksheet. The hierarchical structure of both parts of the XML file is discussed in detail earlier (in Section 3.2.2).

A new XML file needs to be created as the first step. Then, the computed data needs to be written into the two parts of the XML. These steps can be performed by using a library named “XmlWriter”. “XMLWriter” is a library which is used to generate XML documents from C# programming. XML file consists of elements and attributes. An element named “StandardCrashReport” is created. This element contains the two parts of the XML file. The “StandardCrashReport” element acts as parent element for both the parts. Elements like “Summary” and “SimDevice” are created using “XmlWriter.WriteStartElement”.

The simulation device, which is the first part is written to the  XML file, after the file is created. Each simulation device starts with the tag name “SimDevice”, and has an attribute which can store the name of the simulation device. Following that, the data is stored in a hierarchical manner. Along with this, elements where yellow cell color needs to be filled is added as an attribute.

Following the simulation device part, is the summary part. This part follows the hierarchical order of crash, simulation device, and then deployment values. All the crashes are enclosed by a tag named “Summary”. The creation of this “Summary” element in the XML file, and the other elements inside it can be seen in Figure 4.12.



```
// Create a "Summary" tag in the xml file
xmlwriter.WriteStartElement("Summary");
xmlwriter.WriteAttributeString("ParamName", paramfilename);
int count = 0;
for (int j = 0; j < crashid.Count; j++)
{
    xmlwriter.WriteStartElement("Crash");
    xmlwriter.WriteAttributeString("CrashName", crashid[j].ToString());
    xmlwriter.WriteAttributeString("CrashInfo", crashinfo[j].ToString());
    xmlwriter.WriteAttributeString("AssignmentComment", assigncomment[j].ToString());

    for (int i = 0; i < distinct_simdevs.Count; i++)
    {
        xmlwriter.WriteStartElement("SimDevice");
        xmlwriter.WriteAttributeString("SimName", distinct_simdevs[i].ToString());
        xmlwriter.WriteAttributeString("Variationrange", amplituderange);

        xmlwriter.WriteStartElement("SummaryTMax");
        xmlwriter.WriteValue(summary_TMax[count]);
        xmlwriter.WriteEndElement();

        xmlwriter.WriteStartElement("Nom_TMax");
        xmlwriter.WriteValue(summary_Nom[count]);
        xmlwriter.WriteEndElement();
```

**Figure 4.12: Code Snippet to Write to an XML File**

The class “Standard_XmlWriter.cs” writes all the extracted data to the XML file. This XML file is combine with the StyleSheet to generate the Standard Customer Report. This StyleSheet contains all the format options required to convert the XML file into an Excel file. The StyleSheet is specifically created for Standard Customer Report generation. The generated XML contains two parts for simulation device woksheets and summary worksheet. In the same way, the StyleSheet also contains two parts. Along with the two parts, the StyleSheet has a separate section for storing the required styles.

Each style defined in the styles section contains a unique “StyleID”. The “StyleID” is used for elements of the XML file to apply formatting options such as having a different font, adding color to the text, and borders to the cell. The formatting function of adding colors requires conditional checks. The attributes in the XML file have a command whether color should be added or not. These attributes need to be checked and then color needs to be added to the required field.

To perform condition checks in the StyleSheet, elements such as “xsl:choose”, “xsl:when” and “xsl:otherwise” are used. The “xsl:choose” element is used to notify the system that a conditional test needs to be performed. Then, using “xsl:when”, the conditional test is performed. If the condition is satisfied, the code that follows will be executed. When the condition fails, the code which is present in the “xsl:otherwise” section is executed. These three elements are used in the StyleSheet to add colors and perform the required conditional tests.

There are multiple user required formatting options which are to be present in the final output. Formatting options which are required for the simulation device worksheets are as follows:

- A different worksheet should be created for each simulation device. Each worksheet should be named after the simulation device.
- The first cell in the Excel file must contain the name of the simulation device used. This field should have the text in bold font.
- The first column contains crash names followed by their crash information. The crash names must be in bold font.
- The second column in the Excel file must contain assignment comments of the respective crash.
- Tmin, Tmax, Rmin, Rmax and ALL columns for each amplitude step.
- The worksheet must contain borders as per the user requirement. There should also be yellow color present for the required fields.
- Header and Column fixation needs to be done. The width of the columns in the Excel file must be adjusted automatically.

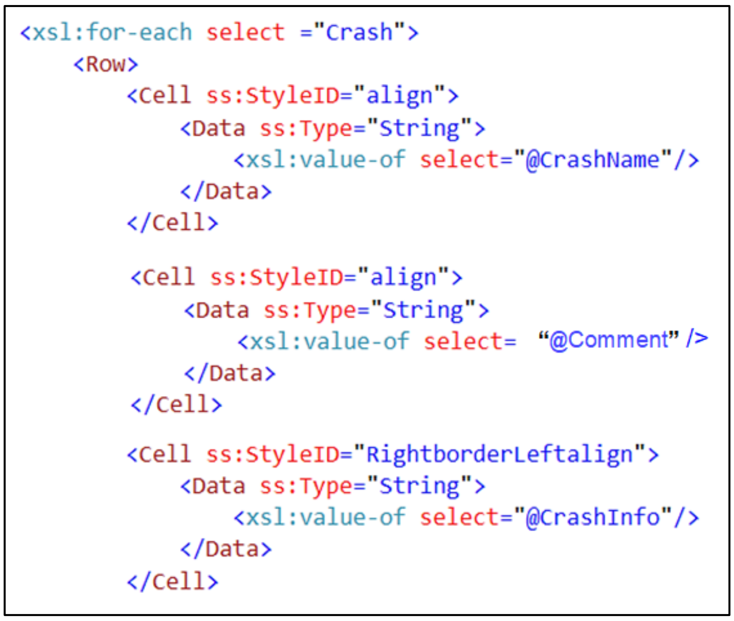
These are the requirements of the simulation device worksheets in Excel file. These formatting options need to be present for the simulation device worksheet. The StyleSheet for this section is designed based on these requirements.
The formatting options required for summary worksheet are as follows:

- The summary worksheet must be the first worksheet of the Excel file.
- The first cell of the summary worksheet must contain the parameter file name.

- The first three columns of this worksheet must be crash code, comment and crash information respectively.
- Each simulation device and its entire amplitude range must be present. This part must contain columns for requirement, max, nominal Tmax, min, and fire behavior.
- The worksheet must contain borders as per user requirement. It should contain yellow color for the required fields.
- Header and Column fixation needs to be done. The width of the columns in the Excel file must be adjusted automatically.

These are the requirements by the user for the summary worksheet.

After considering these requirements, the StyleSheet is designed accordingly. A part of the StyleSheet used to generate the Excel file from XML can be seen in Figure 4.13. This part of the StyleSheet writes the crash name, comment and crash information to the summary worksheet of the Excel file. This is done as per user requirements. The data for crash name and comment must be aligned to the left of the cell. This is done by creating a "StyleID" named "align". The crash information should also be aligned to the left and it must have a border to its right. So, another style is created named as "Right border Left align". Appropriate naming of the styles is important for developing clean, re-usable code. [30]



```
<xsl:for-each select ="Crash">
    <Row>
        <Cell ss:StyleID="align">
            <Data ss:Type="String">
                <xsl:value-of select="@CrashName"/>
            </Data>
        </Cell>

        <Cell ss:StyleID="align">
            <Data ss:Type="String">
                <xsl:value-of select=  “@Comment” />
            </Data>
        </Cell>

        <Cell ss:StyleID="RightborderLeftalign">
            <Data ss:Type="String">
                <xsl:value-of select="@CrashInfo"/>
            </Data>
        </Cell>
```

**Figure 4.13: Screenshot of the XSLT StyleSheet**

The StyleSheet is developed and the required formatting options are enabled as per the user requirements. Most of the styling is performed using StyleSheets. C# is used for some of the styling requirements, such as header and column fixation, and adjusting the width of columns. Along with these options, the re-arranging of the summary worksheet is done using C#. The summary worksheet must be the first worksheet, so that the user can get an overview with the summary worksheet, and then go through the simulation device worksheets. This is done to increase the performance, and keep the StyleSheet simple. The last step is to perform the XSLT conversion of XML, and the developed StyleSheet. This step would result in the generation of the Standard Customer Report in Excel file format.

The last step of converting the XML and StyleSheet starts with loading the StyleSheet. The StyleSheet is parsed alongside the XML file. The “XslCompiledTransform” is a library in C# to perform XSLT conversion. The StyleSheet is loaded by using the “Load()” method. Then, the XML document is combined with the loaded stylesheet. Then, “Transform()” method is used to generate the Excel file. During the conversion, a set of processing instructions are provided to the XSLT. The processing instructions consist of providing the output file format.

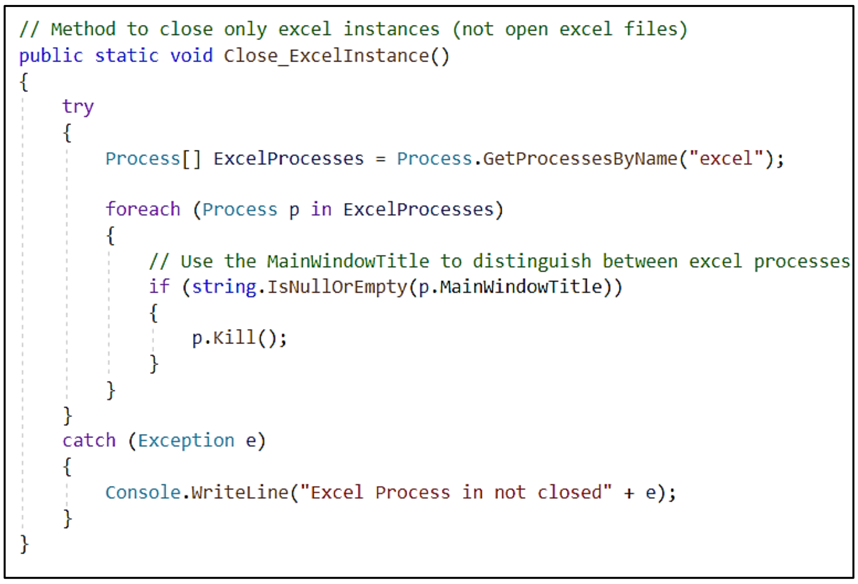
During the conversion process, an Excel instance is created in the background. If an Excel instance is opened in the background, and the user tries to run RSDBnext tool, errors may occur. Multiple background Excel instances can cause memory consumption, and reduce the speed of the system. Hence, the issue with Excel instance needs to be handled when the user runs the RSDBnext tool to generate Standard Customer Reports. To overcome this problem, one of the possible solutions needs to be implemented. The possible options are :

1) Closing the Excel instances automatically each time the tool is started
2) Displaying a message to the user to close the Excel instance manually

The first option is to close the Excel instance automatically. Each time the RSDBnext tool is started for generating Standard Customer Reports, the Excel instances are closed automatically. Using the second option, the user needs to perform additional clicks, such as opening the task manager, and then closing the background Excel instances. After closing the instances, the RSDBnext tool must be started again. The second option is complicated for the user. The first option is better than the second option to close Excel instances. Hence, the first option is selected to close the Excel instances automatically.

To close the Excel instances automatically, a method is created, which is named as “Close_ExcelInstance()”. This method is started before the XSLT conversion process. The developed method identifies whether the Excel instance which is open is a background process running or an open Excel file. Only the background process needs to be closed by the method. To ensure that only the background instances are closed, the "MainWindowTitle" is used. If the "MainWindowTitle" is empty, then this is a background Excel instance. These background instances are then closed automatically in C#. The user doesn't need to do any additional steps. Therefore, by performing the closing of background Excel instances, the performance of the tool is maintained. Figure 4.14 depicts the method to close the background Excel instances. In this method, only the background instances are closed, and not the open Excel files.

For e.g., when the RSDBnext tool is running, if the user stops running the program during this time, an Excel instance is kept open in the background. The "Close_ExcelInstance()" method closes the Excel instance automatically when the user starts the RSDBnext tool again. There will not be any error due to the background Excel instances using this method. The user is also not required to take any extra steps to resolve the problem. Hence, this method of closing Excel instances is efficient. It also maintains the performance and speed of the tool.



```csharp
// Method to close only excel instances (not open excel files)
public static void Close_ExcelInstance()
{
    try
    {
        Process[] ExcelProcesses = Process.GetProcessesByName("excel");

        foreach (Process p in ExcelProcesses)
        {
            // Use the MainWindowTitle to distinguish between excel processes
            if (string.IsNullOrEmpty(p.MainWindowTitle))
            {
                p.Kill();
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("Excel Process in not closed" + e);
    }
}
```

**Figure 4.14: Method in C# to Close Background Excel Instances**

The Excel instances need to be handled for the generation of Standard Customer Report. The developed method to close the Excel instances is presented. These are all the steps implemented to generate the Standard Customer Reports in an efficient manner.

## 4.4 Summary

This chapter discusses the implementation of the RSDBnext tool. The RSDBnext tool is further developed to generate Standard Customer Reports. The input format option to start generating the required report is “Standard”. Along with the format option, the input airbag simulation results is required. These simulation results are extracted from the result file of the RSDBnext tool. This file is generated from the format option, which is “Result”.

The second part discusses the calculations done on the extracted crash data. This section is divided into two sub-sections. The first sub-section contains the checks performed on the input for equal amplitude steps. The next section contains the fire behavior of each crash, and its simulation device.

The last section contains the generation of the required Standard Customer Report along with the XML file. The user requirements for the simulation device worksheet and the summary worksheet are explained. Following that, the StyleSheet is implemented based on these specifications. Then, the process of generating the Standard Customer Report in Excel is explained. The issue with Excel instances is also discussed. The possible solutions to solve this issue is discussed, and the most suitable one is selected to generate the Standard Customer Reports. The results and analysis of the thesis is discussed in the next chapter.

# 5 Results and Analysis

This chapter discusses the results which are obtained by using RSDBnext tool to generate the Standard Customer Reports. In the first section of this chapter, the output worksheets of the simulation devices and summary are presented. These results to generate Standard Customer Reports are compared against the previous methodology. The implemented tool is tested using various testing techniques such as unit testing, white box testing and black box testing. Testing the implemented tool is carried out at different stages of implementation.

Unit testing is performed on all the developed methods in the code. Each developed module of the RSDBnext tool is tested. These code components are examined to discover any errors that may be present. Unit testing helps improve the quality of the code and increase the efficiency of the development process. For e.g., the "Close_ExcelInstance ()" method which is developed to close the background Excel instances is tested individually. It is tested to verify the functionality. A few background Excel instances were generated. Then the RSDBnext tool was started. The Excel instances were closed automatically and the required reports were generated. Likewise, all the methods and classes were tested to verify their functionality during the development of the tool.

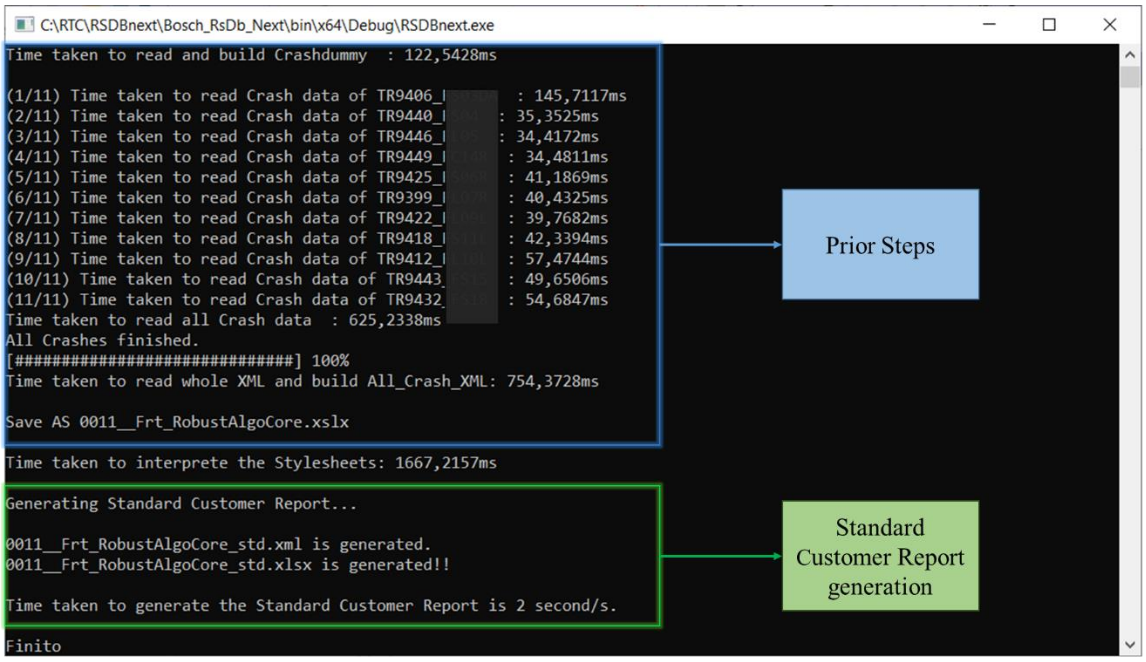
After the complete development of the RSDBnext tool, white box testing and black box testing is carried out. White box testing involved checking and examining the internal structure of the developed code. The code is analyzed to understand its architecture, design, and implementation. Then, the entire code is tested. The airbag application tool is tested to check if there are any errors within the developed methods or classes. The detected errors are fixed and the overall code is improved. [31]

The final test which is performed on the developed tool is the black box testing. It involves using different input databases to evaluate the output reports. The database contains airbag simulated results of different sets of crashes. The different set of crashes may include data from different customers. The RSDBnext tool is tested by generating Standard Customer Reports with different inputs. The generated reports must be the same as generated from the existing approach. Standard Customer Reports were generated from the RSDBnext tool and also the existing approach. The output results showed that the generated reports were identical, indicating that the tool performed as expected. The black box testing process was important to ensure the reliability of the developed RSDBnext tool.

The final output is evaluated based on the generated Standard Customer Report. The aim is to generate the required Standard Customer Reports, as per user requirements. The process to generate the reports must be efficient, and should happen in a faster manner. First, the extraction of required data from the airbag simulation results must be performed efficiently. Then this data needs to be modified and reports should be generated in a stable way such that the system doesn't crash or freeze altogether. The time taken to generate the required reports must be lesser than the existing approach.

The RSDBnext tool is further developed for the report generation process. The tool is developed in C# programming. It is used to extract the required crash data and generate reports. The user of the tool provides the required inputs to the tool. The inputs provided include the format option. To generate the Standard Crash Reports, the user needs to give the format option as "Standard". Once, the required inputs are provided, the tool starts for format option as "Result" and generates the XML file. This XML file is the input for generating the Standard Customer Report. The process steps will be written to a command prompt on the user's screen. The screenshot of the command prompt during the standard crash report generation can be seen in Figure 5.1.

The blue part marked in the figure represents the generation of result XML for generating Standard Customer Report. The used input in the figure contains data for eleven crashes. The simulated data is extracted from the database for all these crashes. Then the result XML is generated by the input as "format = result". This result XML is then taken as the input. The input is then read by the developed tool. When the process starts, an output line is printed to the console. This output lets the user know that the process to generate the required reports has started. This is shown by the green part marked in Figure 5.1. The start of RSDBnext for generating Standard Customer Report starts from extracting the required crash data from the result XML.


C:\RTC\RSDBnext\Bosch_RsDb_Next\bin\x64\Debug\RSDBnext.exe
Prior Steps
Standard Customer Report generation

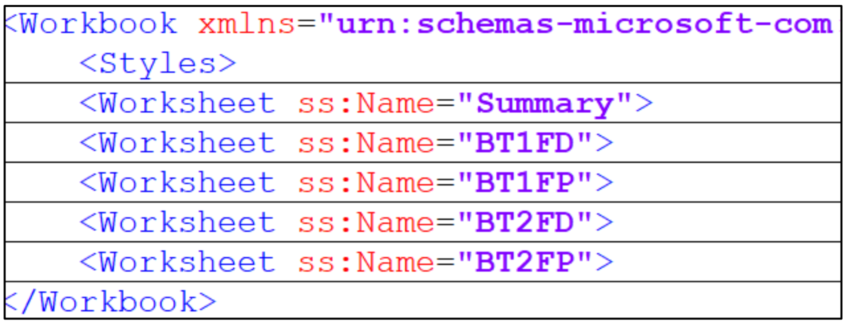
```
Time taken to read and build Crashdummy  : 122,5428ms

(1/11) Time taken to read Crash data of TR9406_I[redacted]  : 145,7117ms
(2/11) Time taken to read Crash data of TR9440_I      : 35,3525ms
(3/11) Time taken to read Crash data of TR9446_I      : 34,4172ms
(4/11) Time taken to read Crash data of TR9449_I       : 34,4811ms
(5/11) Time taken to read Crash data of TR9425_I       : 41,1869ms
(6/11) Time taken to read Crash data of TR9399_I       : 40,4325ms
(7/11) Time taken to read Crash data of TR9422_I       : 39,7682ms
(8/11) Time taken to read Crash data of TR9418_I       : 42,3394ms
(9/11) Time taken to read Crash data of TR9412_I       : 57,4744ms
(10/11) Time taken to read Crash data of TR9443_      : 49,6506ms
(11/11) Time taken to read Crash data of TR9432_      : 54,6847ms
Time taken to read all Crash data  : 625,2338ms
All Crashes finished.
[##############################] 100%
Time taken to read whole XML and build All_Crash_XML: 754,3728ms

Save AS 0011__Frt_RobustAlgoCore.xslx

Time taken to interprete the Stylesheets: 1667,2157ms

Generating Standard Customer Report...

0011__Frt_RobustAlgoCore_std.xml is generated.
0011__Frt_RobustAlgoCore_std.xlsx is generated!!

Time taken to generate the Standard Customer Report is 2 second/s.

Finito
```

**Figure 5.1: Command Prompt during Standard Report Generation**

Once the required crash data is extracted, the necessary computations are performed on it. Then, a suitable file format is chosen to store the results to an intermediate file. This suitable file format ensures that the reports are generated in a faster and stable manner. The intermediate file format is of XML file type. This is chosen to ensure the data is stored structurally. The computed data is written to XML file format. This output XML file is named after the criterion name and “std” is added at the end. “std” is used to represent that the file is Standard Customer Report. For e.g., if the criterion “Front Crash Set” is used as input, then the output XML file will be named as “Front Crash Set_std.xml”.

The developed XML file contains two parts in it. The first part is developed to store all the data for the simulation device worksheets. The second part is designed to store the required crash information for the summary worksheet. The separation of the XML file to two parts is done to maintain the performance of the overall report generation process. It also keeps the file structure simple. An example output of the generated XML file can be seen in Figure 5.2.

The first part in the generated XML file is simulation device worksheets. The generated XML file in Figure 5.2 contains four simulation devices. The four simulation devices in the XML file are used to generate four worksheets in the output Standard Customer Report. The second part in the generated XML file is the summary part. This is placed at the beginning of the XML file, as the output Excel file must have the first worksheet as the summary worksheet. The Excel worksheets present are multiple simulation device sheets and summary sheet. The number of simulation devices varies depending on the airbag simulation results. To ease viewing the results for the user, it is a requirement for the summary sheet to be placed as the first worksheet in the generated Standard Customer Report.



```
<Workbook xmlns="urn:schemas-microsoft-com
    <Styles>
    <Worksheet ss:Name="Summary">
    <Worksheet ss:Name="BT1FD">
    <Worksheet ss:Name="BT1FP">
    <Worksheet ss:Name="BT2FD">
    <Worksheet ss:Name="BT2FP">
</Workbook>
```

**Figure 5.2: Generated Standard Customer Report XML File**

The generated Standard Customer Report XML file is combined with the developed StyleSheet. This StyleSheet contains the required styles and formatting rules to get the output as per the user requirements. The use of StyleSheet is done for customization of the output, such as style, size, and color schemes. The generated output XML file is combined with the developed StyleSheet. This combination undergoes a conversion process to produce the final output. The conversion is done using XSLT. This ensures that the developed output reports have consistent style and layout applied to them. The StyleSheet is developed to produce correct formatting of the reports irrespective of the input. The final output from the RSDBnext tool is in Excel file format.

The Excel file format is chosen because of its ability to handle multiple worksheets efficiently. The generated report must contain all the simulation device worksheets and the summary worksheets. The output results which are the Standard Customer Reports are discussed further.

## 5.1 Generation of Standard Customer Reports in Excel

The Standard Crash Reports are generated in Excel file format. This is obtained from the XML and XSLT conversion process. A specific StyleSheet for the required formatting options is designed for the conversion process. The Excel file is composed of two parts which are discussed in the further sub-sections. The simulation devices worksheets are discussed first. Then, the summary worksheet in the Excel file is discussed.

### 5.1.1 Simulation Device Worksheets

In this sub-section, the simulation device worksheets which are generated in the Excel file are examined. These worksheets are developed as per user requirements. According to the user requirements, the data of each simulation device must be stored in one worksheet. The simulation device worksheets in the Excel file can be seen in Figure 5.3. The example output in the figure has four simulation devices which can be seen.

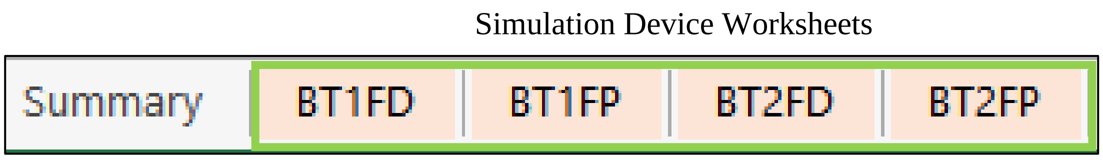
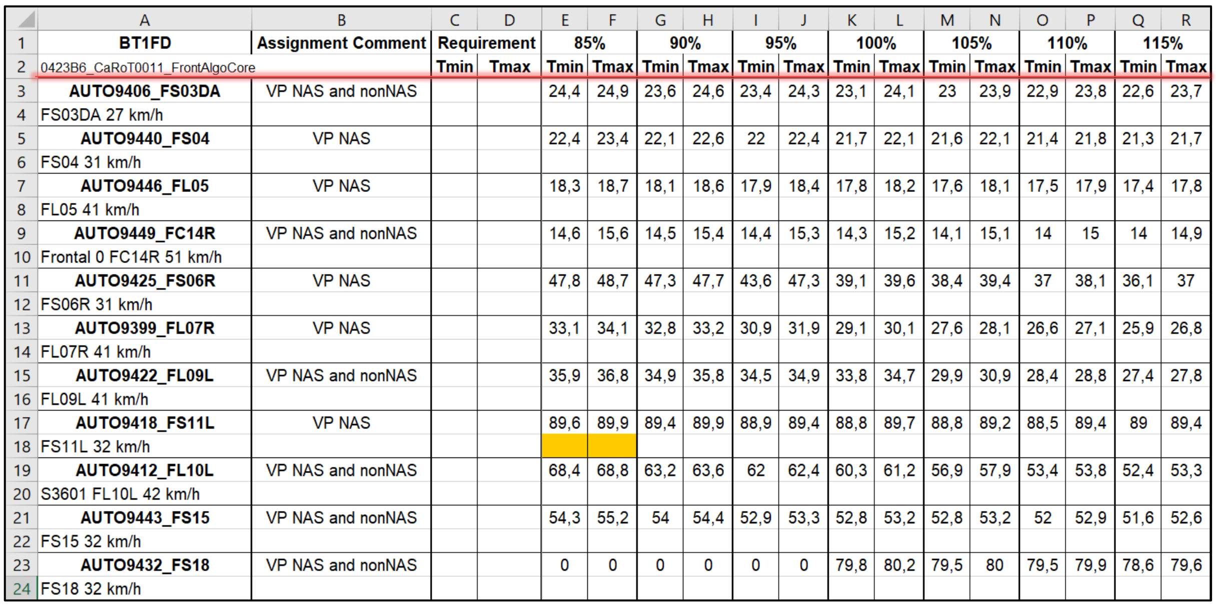


**Figure 5.3: Simulation Device Worksheets in the Excel file**

The crash data stored in the simulation device worksheet is based on user requirements. The first cell in the Excel file contains the simulation device's name. This is shown in Figure 5.4. This name should be in bold font. The main header section contains fields such as comment, requirement, amplitude steps, and "ALL". The "ALL" section in the worksheet represents that an airbag was deployed for a particular amplitude step. There are sub-headers for these main headers. The values to these sub-headers are computed. The sub-headers are Tmin, Tmax, Rmin, Rmax, and the amplitude steps for a crash.



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **BT1FD** | **Assignment Comment** | **Requirement** | | **85%** | | **90%** | | **95%** | | **100%** | | **105%** | | **110%** | | **115%** | |
| 2 | 0423B6_CaRoT0011_FrontAlgoCore | | **Tmin** | **Tmax** | **Tmin** | **Tmax** | **Tmin** | **Tmax** | **Tmin** | **Tmax** | **Tmin** | **Tmax** | **Tmin** | **Tmax** | **Tmin** | **Tmax** | **Tmin** | **Tmax** |
| 3 | **AUTO9406_FS03DA** | VP NAS and nonNAS | | | 24,4 | 24,9 | 23,6 | 24,6 | 23,4 | 24,3 | 23,1 | 24,1 | 23 | 23,9 | 22,9 | 23,8 | 22,6 | 23,7 |
| 4 | FS03DA 27 km/h | | | | | | | | | | | | | | | | | |
| 5 | **AUTO9440_FS04** | VP NAS | | | 22,4 | 23,4 | 22,1 | 22,6 | 22 | 22,4 | 21,7 | 22,1 | 21,6 | 22,1 | 21,4 | 21,8 | 21,3 | 21,7 |
| 6 | FS04 31 km/h | | | | | | | | | | | | | | | | | |
| 7 | **AUTO9446_FL05** | VP NAS | | | 18,3 | 18,7 | 18,1 | 18,6 | 17,9 | 18,4 | 17,8 | 18,2 | 17,6 | 18,1 | 17,5 | 17,9 | 17,4 | 17,8 |
| 8 | FL05 41 km/h | | | | | | | | | | | | | | | | | |
| 9 | **AUTO9449_FC14R** | VP NAS and nonNAS | | | 14,6 | 15,6 | 14,5 | 15,4 | 14,4 | 15,3 | 14,3 | 15,2 | 14,1 | 15,1 | 14 | 15 | 14 | 14,9 |
| 10 | Frontal 0 FC14R 51 km/h | | | | | | | | | | | | | | | | | |
| 11 | **AUTO9425_FS06R** | VP NAS | | | 47,8 | 48,7 | 47,3 | 47,7 | 43,6 | 47,3 | 39,1 | 39,6 | 38,4 | 39,4 | 37 | 38,1 | 36,1 | 37 |
| 12 | FS06R 31 km/h | | | | | | | | | | | | | | | | | |
| 13 | **AUTO9399_FL07R** | VP NAS | | | 33,1 | 34,1 | 32,8 | 33,2 | 30,9 | 31,9 | 29,1 | 30,1 | 27,6 | 28,1 | 26,6 | 27,1 | 25,9 | 26,8 |
| 14 | FL07R 41 km/h | | | | | | | | | | | | | | | | | |
| 15 | **AUTO9422_FL09L** | VP NAS and nonNAS | | | 35,9 | 36,8 | 34,9 | 35,8 | 34,5 | 34,9 | 33,8 | 34,7 | 29,9 | 30,9 | 28,4 | 28,8 | 27,4 | 27,8 |
| 16 | FL09L 41 km/h | | | | | | | | | | | | | | | | | |
| 17 | **AUTO9418_FS11L** | VP NAS | | | 89,6 | 89,9 | 89,4 | 89,9 | 88,9 | 89,4 | 88,8 | 89,7 | 88,8 | 89,2 | 88,5 | 89,4 | 89 | 89,4 |
| 18 | FS11L 32 km/h | | | | | | | | | | | | | | | | | |
| 19 | **AUTO9412_FL10L** | VP NAS and nonNAS | | | 68,4 | 68,8 | 63,2 | 63,6 | 62 | 62,4 | 60,3 | 61,2 | 56,9 | 57,9 | 53,4 | 53,8 | 52,4 | 53,3 |
| 20 | S3601 FL10L 42 km/h | | | | | | | | | | | | | | | | | |
| 21 | **AUTO9443_FS15** | VP NAS and nonNAS | | | 54,3 | 55,2 | 54 | 54,4 | 52,9 | 53,3 | 52,8 | 53,2 | 52,8 | 53,2 | 52 | 52,9 | 51,6 | 52,6 |
| 22 | FS15 32 km/h | | | | | | | | | | | | | | | | | |
| 23 | **AUTO9432_FS18** | VP NAS and nonNAS | | | 0 | 0 | 0 | 0 | 0 | 0 | 79,8 | 80,2 | 79,5 | 80 | 79,5 | 79,9 | 78,6 | 79,6 |
| 24 | FS18 32 km/h | | | | | | | | | | | | | | | | | |

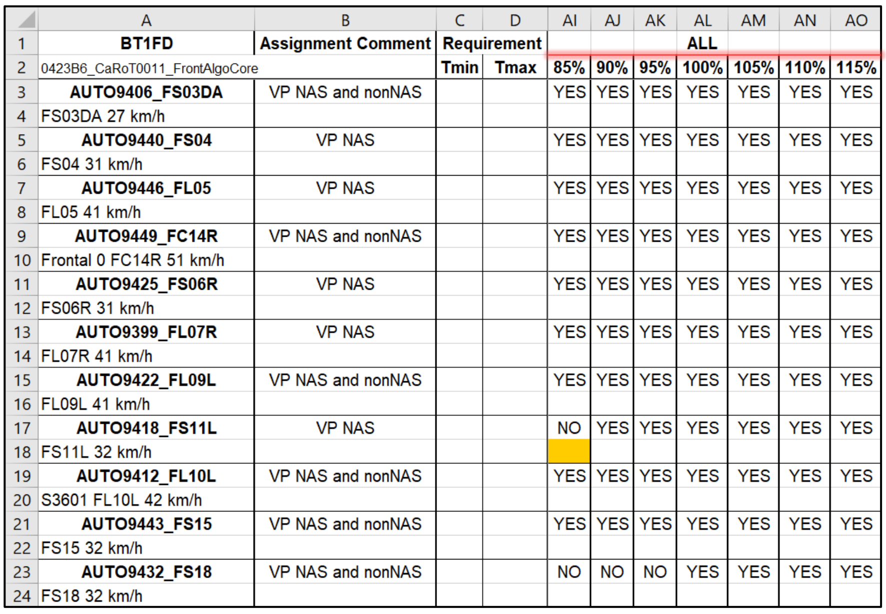
**Figure 5.4: Screenshot of the Simulation Device Worksheet**

The field below the simulation device name should contain the parameter file name. This is extracted from the input file. The formatting requirement for this field is that the font size should be smaller than the other fields. The next data which is displayed in the output file is the crash code and crash information. The data of each crash is present in two rows. The first row contains information such as crash code, comment, and deployment values. The second row contains crash information, with background color based on the requirements for each crash. The crash codes should be highlighted by making the font of the text bold, and centre aligned. The required formatting is done by using the developed StyleSheet. The crash data should be stored correctly in the report. These reports must be readable by the user directly. Hence, the formatting of the output file is necessary.

The airbag deployment values are rounded upto one digit as per requirements. The values such as the minimum fire time and maximum fire time is stored in their respective columns. The computed "ALL" values are which are stored in the XML file are converted to the output file. For each crash and amplitude step, the values of "ALL" are stored in the output Excel file.

This output file contains the airbag fire times for a particular simulation device. The crashes are on the left side of the worksheet, and the deployment times of the simulation device on the right. When the airbag deployment value for a crash is zero, it means that the airbag did not fire for that particular amplitude step. This is about the simulation device worksheet present in the Standard Crash Report.

In Figure 5.5, the "ALL" section in the worksheet contains either a "YES" or "NO". This depicts whether the airbag has fired or not for that particular amplitude step. When the field contains a "YES", it means the airbag has deployed. If any of the field contains a "NO", then it means the airbag has not deployed for all time variations. For some cases, there is a yellow marking below the "NO" field.  If there is a yellow marking below the field, then it indicates that there was a partial deployment for that case. When the airbag deployment probability is not 100%, it is called partial deployment. This yellow marking helps the customer to identify the partial deployment cases easily.



| | A | B | C | D | AI | AJ | AK | AL | AM | AN | AO |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | BT1FD | Assignment Comment | Requirement | | | | | ALL | | | |
| 2 | 0423B6_CaRoT0011_FrontAlgoCore | | Tmin | Tmax | 85% | 90% | 95% | 100% | 105% | 110% | 115% |
| 3 | AUTO9406_FS03DA | VP NAS and nonNAS | | | YES | YES | YES | YES | YES | YES | YES |
| 4 | FS03DA 27 km/h | | | | | | | | | | |
| 5 | AUTO9440_FS04 | VP NAS | | | YES | YES | YES | YES | YES | YES | YES |
| 6 | FS04 31 km/h | | | | | | | | | | |
| 7 | AUTO9446_FL05 | VP NAS | | | YES | YES | YES | YES | YES | YES | YES |
| 8 | FL05 41 km/h | | | | | | | | | | |
| 9 | AUTO9449_FC14R | VP NAS and nonNAS | | | YES | YES | YES | YES | YES | YES | YES |
| 10 | Frontal 0 FC14R 51 km/h | | | | | | | | | | |
| 11 | AUTO9425_FS06R | VP NAS | | | YES | YES | YES | YES | YES | YES | YES |
| 12 | FS06R 31 km/h | | | | | | | | | | |
| 13 | AUTO9399_FL07R | VP NAS | | | YES | YES | YES | YES | YES | YES | YES |
| 14 | FL07R 41 km/h | | | | | | | | | | |
| 15 | AUTO9422_FL09L | VP NAS and nonNAS | | | YES | YES | YES | YES | YES | YES | YES |
| 16 | FL09L 41 km/h | | | | | | | | | | |
| 17 | AUTO9418_FS11L | VP NAS | | | NO | YES | YES | YES | YES | YES | YES |
| 18 | FS11L 32 km/h | | | | 🟨 | | | | | | |
| 19 | AUTO9412_FL10L | VP NAS and nonNAS | | | YES | YES | YES | YES | YES | YES | YES |
| 20 | S3601 FL10L 42 km/h | | | | | | | | | | |
| 21 | AUTO9443_FS15 | VP NAS and nonNAS | | | YES | YES | YES | YES | YES | YES | YES |
| 22 | FS15 32 km/h | | | | | | | | | | |
| 23 | AUTO9432_FS18 | VP NAS and nonNAS | | | NO | NO | NO | YES | YES | YES | YES |
| 24 | FS18 32 km/h | | | | | | | | | | |

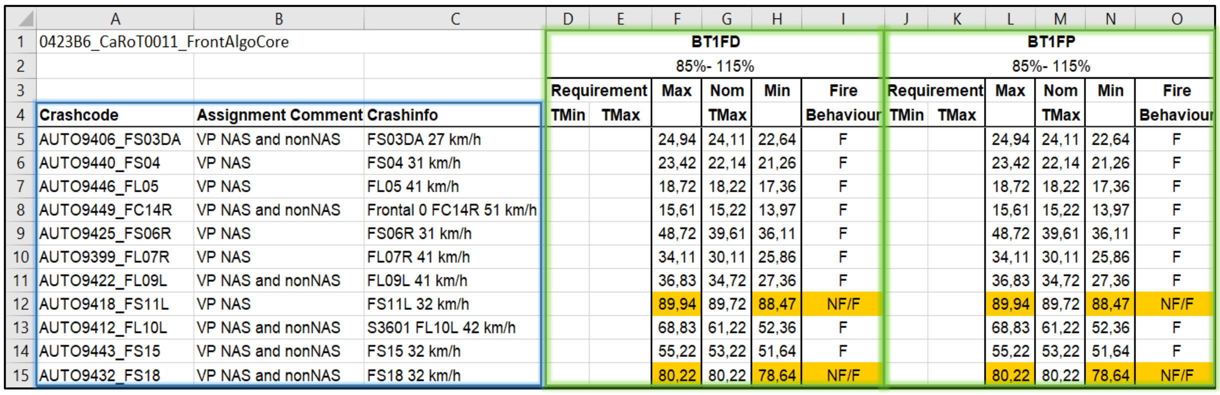
**Figure 5.5: Screenshot of "ALL" Section in the Simulation Device Worksheet**

An overview of the deployment times is provided by the "ALL" part in the Excel worksheet. The worksheet depicted is for a simulation device. Similarly, there can be numerous simulation devices which have the same structure. The summary worksheet, which gives an overview of all the simulation devices is discussed further.

### *5.1.2* Summary Worksheet

This sub-section discusses the summary worksheet generated in the Excel file. The summary worksheet has a different structure compared to the simulation device worksheets. An overview of the crash simulated data is present in the summary worksheet. The first field in this worksheet is the parameter file name. Then, the first three columns of the worksheet are crash code, comment and crash information. The header for the file is the simulation device name, and following it is the amplitude variation range. There are further sub-headings in the Standard Customer Report, and they are requirement, max, nom max, min, and fire behaviour. Each of these sub-headings are an important part of the results, and are filled in based on the computed data.

The requirement fields are added separately by the user, as they are provided as inputs from the customer. Figure 5.6 shows a screenshot of the summary worksheet in the Excel file. The max field contains the maximum of the Tmax values for the entire amplitude range. The nominal Tmax contains the Tmax value at 100% amplitude step. The min field contains the minimum value of Tmin values. This computed data is stored in the internal structure first. Then, this computed data is written to XML file. This data from the XML file is transferred to the Excel file. In the XML file, the required data is stored in a separate section named summary.



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0423B6_CaRoT0011_FrontAlgoCore | | | BT1FD | | | | | | BT1FP | | | | | |
| 2 | | | | 85%- 115% | | | | | | 85%- 115% | | | | | |
| 3 | | | | Requirement | | Max | Nom | Min | Fire | Requirement | | Max | Nom | Min | Fire |
| 4 | Crashcode | Assignment Comment | Crashinfo | TMin | TMax | | TMax | | Behaviour | TMin | TMax | | TMax | | Behaviour |
| 5 | AUTO9406_FS03DA | VP NAS and nonNAS | FS03DA 27 km/h | | | 24,94 | 24,11 | 22,64 | F | | | 24,94 | 24,11 | 22,64 | F |
| 6 | AUTO9440_FS04 | VP NAS | FS04 31 km/h | | | 23,42 | 22,14 | 21,26 | F | | | 23,42 | 22,14 | 21,26 | F |
| 7 | AUTO9446_FL05 | VP NAS | FL05 41 km/h | | | 18,72 | 18,22 | 17,36 | F | | | 18,72 | 18,22 | 17,36 | F |
| 8 | AUTO9449_FC14R | VP NAS and nonNAS | Frontal 0 FC14R 51 km/h | | | 15,61 | 15,22 | 13,97 | F | | | 15,61 | 15,22 | 13,97 | F |
| 9 | AUTO9425_FS06R | VP NAS | FS06R 31 km/h | | | 48,72 | 39,61 | 36,11 | F | | | 48,72 | 39,61 | 36,11 | F |
| 10 | AUTO9399_FL07R | VP NAS | FL07R 41 km/h | | | 34,11 | 30,11 | 25,86 | F | | | 34,11 | 30,11 | 25,86 | F |
| 11 | AUTO9422_FL09L | VP NAS and nonNAS | FL09L 41 km/h | | | 36,83 | 34,72 | 27,36 | F | | | 36,83 | 34,72 | 27,36 | F |
| 12 | AUTO9418_FS11L | VP NAS | FS11L 32 km/h | | | 89,94 | 89,72 | 88,47 | NF/F | | | 89,94 | 89,72 | 88,47 | NF/F |
| 13 | AUTO9412_FL10L | VP NAS and nonNAS | S3601 FL10L 42 km/h | | | 68,83 | 61,22 | 52,36 | F | | | 68,83 | 61,22 | 52,36 | F |
| 14 | AUTO9443_FS15 | VP NAS and nonNAS | FS15 32 km/h | | | 55,22 | 53,22 | 51,64 | F | | | 55,22 | 53,22 | 51,64 | F |
| 15 | AUTO9432_FS18 | VP NAS and nonNAS | FS18 32 km/h | | | 80,22 | 80,22 | 78,64 | NF/F | | | 80,22 | 80,22 | 78,64 | NF/F |

**Figure 5.6: Snapshot of a Summary Worksheet**

The fire behaviour value present in this worksheet can be amongst one of the three possibilities. The first possibility is full fire, which means that airbag deployment is 100%. This is abbreviated as "F". It means that for all amplitude steps the value is a "YES". If the case is the airbag not deploying for all amplitude variations, then the second possibility is chosen. It is "No Fire", which means that airbag deployment is 0%. This is abbreviated as "NF". The maximum and minimum fields remain zero during this case. This must be present in the output Excel report according to the requirements of the user. When there is partial deployment, it means that there are a few "YES" and "NO" values. For this particular case, the cell-value is marked with "NF/F". The partial deployment fields should be marked with yellow background color. Along with the marking of this field, the max and min field should also be marked yellow for this case with partial deployment. For the amplitude step at 100%, if there is partial deployment at that particular case, then the field is marked with yellow background color.

In Figure 5.6, there is a summary of two simulation devices. This summary sheet contains an overview of all the simulation devices in the worksheet.

The Standard Customer Report which is the Excel file, is used by the customer. Hence, it should be formatted according to the user requirements. An example of the output Standard Customer Report is discussed in this section. An efficient methodology was developed to generate Standard Customer Reports. The evaluation of the developed methodology is done against the existing method to generate the required reports in the following section.

## 5.2 Evaluation Against the Previous Method

In this section, the developed tool is evaluated against the existing methodology to generate Standard Customer Reports. Firstly, the existing solution uses Excel macros which is solw and complex. The developed methodology must generate the required reports in an efficient manner.

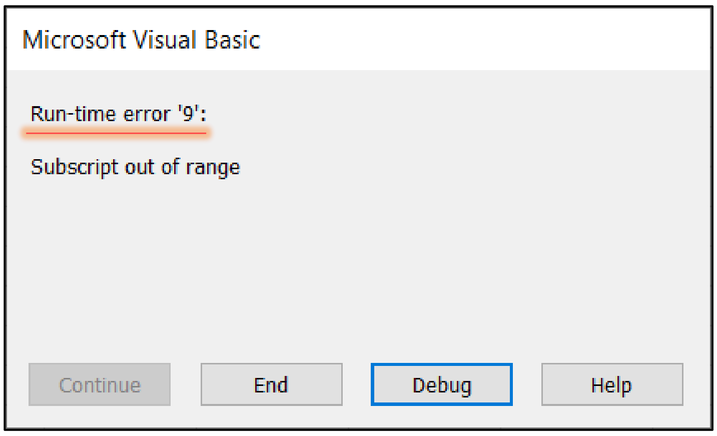
RSDBnext tool is developed to extract the required input data from airbag simulation results. This extracted data is stored internally for performing the required computations. Then, the crash data is computed and stored into an XML file. This process of storing the computed values to an XML file is done to ensure the stability of the entire process. The generated XML file is transformed to an Excel file after applying the required format options. This conversion process makes sure that the generated Standard Customer Report is according to the user requirements. This is verified by comparing the existing solution's output with the developed solution's output. The final Excel files are the same. Hence, the Standard Customer Reports are generated as per user requirements.

The developed RSDBnext tool is implemented with additional features such as automatic adjustment of column width and column and row fixation. The Excel file contains multiple number of columns and rows in each worksheet. The header and the crash codes must be visible when the user wants to see the "ALL" section, which is at the right end of the Excel file. To make this possible, the header and column fixation feature is added in the Excel file. The previous solution did not have merged cells for some header fields. The Excel macro took a lot of time to perform this operation and the system became unstable. So, this merging option was not added in the previous solution. In the developed tool, the merging of cells is possible. In the output from the previous method, the fire behavior header field was left empty. This caused ambiguity for the user. In the developed RSDBnext tool, this header field was added to make sure that the columns are named correctly. Moreover, the summary worksheet didn't have a finishing border line. This border line was implemented in the developed solution. This is done to indicate the last crash in the worksheet to the user. These options help to have the crash information in an organized manner in the Standard Customer Report.

The developed RSDBnext tool is successfully integrated into the existing toolchain. The functionality of the developed RSDBnext tool with the toolchain is tested and verified under various test conditions. The developed tool works well with the current toolchain. This developed methodology doesn't use Excel macros to maintain the performance of the report generation process.

The user had to perform several clicks to generate the reports using Excel macros. To create reports, this procedure required a lot of time. Moreover, the steps needed to be performed by the user to start the Excel macros is complex. The user had to learn the clicks required to be done. Then, by performing the clicks The user then had to wait for quite some time to get the desired reports. In the developed methodology, the user does not have to perform many steps. The process is much simpler. The input options have to be provided to the RSDBnext tool, then the reports will be generated. As, the existing tool is further developed, the users are familiar with the tool. The users know the input which needs to be provided to the tool. The only step which has to be done by the user is to provide the format option as "Standard". The number of steps the user has to perform using the previous method is seven, whereas only one step needs to be done using the developed tool. Hence, the complexity of generating reports for the user is much lesser than the existing methodology using Excel macros.

The previous methodology using Excel macros could not generate Standard Customer Reports in all cases. There were errors for some unusual scenarios. For e.g., when there was only one simulation device in the airbag simulation results. The existing method showed an error message on the screen and failed to generate the required output. The error displayed by the existing methodology is shown in Figure 5.7. The possible solution to generate the required reports was either by updating the macro or creating a new one. This process was quite tedious. The developed RSDBnext tool handled this issue. This is discussed further.

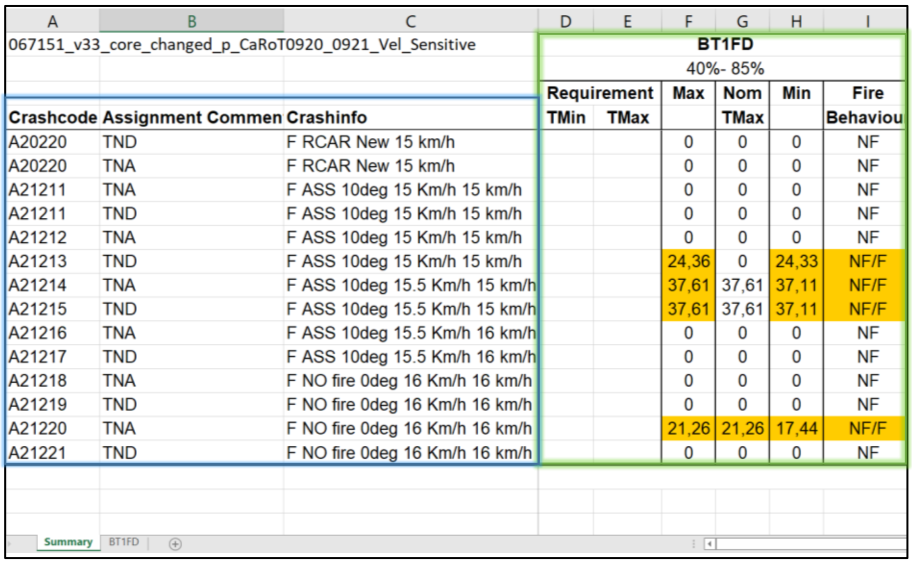

Microsoft Visual Basic

Run-time error '9':

Subscript out of range

**Figure 5.7: Error Message Generated with the Previous Method**

The proposed solution handles the case with one simulation device. It can generate the required reports for any number of simulation devices. Figure 5.8 shows an example of the RSDBnext tool handling this issue. Hence, the developed tool is proven to have better functionality than the previous method.



| A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| 067151_v33_core_changed_p_CaRoT0920_0921_Vel_Sensitive | | | BT1FD | | | | | |
| | | | 40%- 85% | | | | | |
| | | | Requirement | | Max | Nom | Min | Fire |
| Crashcode | Assignment Commen | Crashinfo | TMin | TMax | | TMax | | Behaviou |
| A20220 | TND | F RCAR New 15 km/h | | | 0 | 0 | 0 | NF |
| A20220 | TNA | F RCAR New 15 km/h | | | 0 | 0 | 0 | NF |
| A21211 | TNA | F ASS 10deg 15 Km/h 15 km/h | | | 0 | 0 | 0 | NF |
| A21211 | TND | F ASS 10deg 15 Km/h 15 km/h | | | 0 | 0 | 0 | NF |
| A21212 | TNA | F ASS 10deg 15 Km/h 15 km/h | | | 0 | 0 | 0 | NF |
| A21213 | TND | F ASS 10deg 15 Km/h 15 km/h | | | 24,36 | 0 | 24,33 | NF/F |
| A21214 | TNA | F ASS 10deg 15.5 Km/h 15 km/h | | | 37,61 | 37,61 | 37,11 | NF/F |
| A21215 | TND | F ASS 10deg 15.5 Km/h 15 km/h | | | 37,61 | 37,61 | 37,11 | NF/F |
| A21216 | TNA | F ASS 10deg 15.5 Km/h 16 km/h | | | 0 | 0 | 0 | NF |
| A21217 | TND | F ASS 10deg 15.5 Km/h 16 km/h | | | 0 | 0 | 0 | NF |
| A21218 | TNA | F NO fire 0deg 16 Km/h 16 km/h | | | 0 | 0 | 0 | NF |
| A21219 | TND | F NO fire 0deg 16 Km/h 16 km/h | | | 0 | 0 | 0 | NF |
| A21220 | TNA | F NO fire 0deg 16 Km/h 16 km/h | | | 21,26 | 21,26 | 17,44 | NF/F |
| A21221 | TND | F NO fire 0deg 16 Km/h 16 km/h | | | 0 | 0 | 0 | NF |

**Figure 5.8: Standard Customer Report Containing One Simulation Device**

The developed RSDBnext tool is stable compared to the previous solution. The existing methodology generates reports using C#, XML and XSLT StyleSheet. The handling of the data in the RSDBnext tool is done in an efficient manner. Whereas, the existing methodology completely relied on Excel macros. When the Excel macro was started, many times the system becomes unresponsive and freezes for a long duration. The user cannot work on the main system even. The progress bar displays “Not Responding”, and even the Excel application displays the same. This can be seen in Figure 5.9. These system crashes do not happen with the new approach of RSDBnext tool.

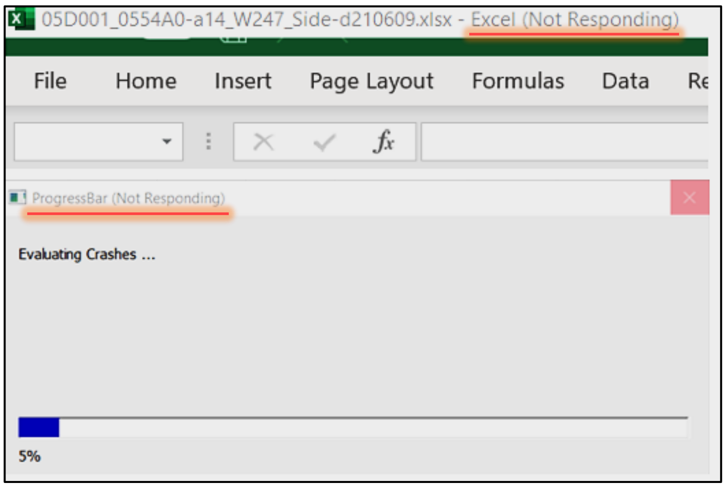

05D001_0554A0-a14_W247_Side-d210609.xlsx - Excel (Not Responding)
File
Home
Insert
Page Layout
Formulas
Data
ProgressBar (Not Responding)
Evaluating Crashes ...
5%

**Figure 5.9: Stability Issues with Excel Macro in the Previous Method**

Most of the system resources are used up by running the Excel macro. This leads to a slower system. The performance of the system is reduced, as it consumes a significant amount of its resources. The resources such as disk space, CPU, and memory are used by the Excel macro. Consuming a lot of memory leads to a slower report generation process. By using a stable methodology like RSDBnext to generate Standard Customer Reports, the performance of the report generation process is maintained.

The previous methodology to generate Standard Customer Report is slow. It takes a long time to generate the expected output. The user has to wait for the reports to be generated. During the report generation time, the user could not use any Microsoft Office products or even the main system. Hence, the Standard Customer Reports needs to be generated in a faster way. The timing factor of the report generation is discussed in Table 5.1.

| Database File Size (in MB) | Time Taken by Previous Approach (in seconds) | Time Taken by RSDBnext Tool (in seconds) |
|---|---|---|
| 23 | 40 | 3 |
| 63.5 | 160 | 8 |
| 501 | 1680 | 78 |

**Table 5.1: Comparison of Previous Method to the Developed Approach**

The generation of Standard Customer Reports using the existing approach consumes a lot of time. As the database file size increases, the time taken increases exponentially. In the third case with a database size of 501 MB, the Excel macro was started for generating the required report. When the existing approach was started for the first time, the screen was unresponsive. When it was started for the second time, then the report was generated after 28 minutes. The time taken by the Excel macro to generate reports was recorded manually using a timer. The overall process to produce the required reports using Excel macros is quite unstable. When the same database was used with the RSDBnext tool, then the time taken was around 1½ minute to generate the report. By analyzing the time take to generate the Standard Customer Reports using both the approaches for various database file sizes, the developed methodology is faster by a factor of around 18.

Therefore, after analyzing the results and evaluation of the RSDB next tool, it can be seen that the main objective of the thesis work is achieved. The required Standard Customer Report generation for airbag simulation results is done in a faster and efficient manner.

## 5.3 Summary

The results and analysis of the developed tool were examined in this chapter. The output results of the developed solution is presented. The simulation device worksheets and the summary worksheet which together form the Standard Customer Report is seen. Then, the developed solution is evaluated against the existing methodology. The RSDBnext tool uses C# programming for handling the airbag simulation results in the tool. The output is generated in a stable way using XML as the intermediate file, and converting it to Excel with the help of StyleSheet. The RSDBnext tool is evaluated against the previous method. According to the results, the developed RSDBnext tool is about 18 times faster than the previous approach. Using RSDBnext to generate Standard Customer Reports is simple, and the overall process is stable. The previous approach was not stable and could not generate reports for all the known cases. The Standard Customer Report generated by the RSDBnext is as per user requirements. The developed approach is evaluated by comparing the old report with the current report. Overall, the proposed solution generates the required report in a faster way. The conclusion and scope for future work will be presented further.

# 6 Conclusion and Future Scope

The main aim of the thesis work is to develop an efficient method to generate Standard Customer Reports for airbag simulation results. The reports are used to present the simulation results to the customer in an efficient way. The existing RSDBnext airbag application tool was further developed to generate the required reports. The previous approach used Excel macros to produce the output. The process to get the output Excel file with the usage of Excel macros was complex and quite unstable. Furthermore, the existing report generation process was slow. Hence, a new tool was developed to generate Standard Customer Reports in an efficient way, by overcoming the drawbacks of the old method.

The generated reports from the RSDBnext tool must be as per user requirements. The input format option provided to the tool is “Standard”. This must be provided by the user to generate the Standard Customer Report. After the input is provided and the tool is started, the airbag simulation results are extracted by C# programming, and are stored in an internal data structure. Then, the required computations are performed on this data. The values are then written to an XML file. XML file is chosen because of it hierarchical structure of storing data. The XML file is combined with a StyleSheet. This combination of XML and XSLT StyleSheet is converted to an output Excel file.

In the output report, data for each simulation device is stored in a different worksheet. A summary sheet consisting of all simulation devices is also present in the output file. This Excel file is the required Standard Customer Report. The developed tool is tested to check for its functionality. The RSDBnext tool is developed to overcome the drawbacks of the existing approach, and generate Standard Customer Reports efficiently.

The future scope of the developed RSDBnext tool could include adding the requirement values to the output. This feature could enable the customer to compare the added requirement values with the values already present in the file. Environments are the different scenarios or conditions possible for an airbag simulation device. The multiple environments could be frontal crash and side crash. The tool can further be developed to store results for multiple environments. A different worksheet could be created for each environment. The developed RSDBnext tool generates reports based on a pre-designed template, which is the StyleSheet. Future developments could include an option for the user to create their own templates. This could enable the user to customize the report. Furthermore, the tool could be integrated with artificial intelligence to help identify any potential issues with the airbag simulation results. For e.g., AI could analyze the airbag simulation data and predict airbag firing times. The entire process could be eased, hence improving the performance of airbag application tools. This could in turn enhance airbag systems.

# List of References

[1] Y. Cheng, Y. Li and C. Yang, "The Design of an Airbag Automatic Inflator and the Simulation Analysis of Airbag in the Unfolding Process," *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, 2020, pp. 961-965.

[2] Robert Bosch GmbH 2023 Web Page [Online], Available: https://www.bosch.com/stories/history-of-airbag-control-units. Accessed: March 5, 2023.

[3] M. Lehmann. (2023). Student Page, CC - OSS in the Bosch Docupedia Page (internal bosch website) [Online]. Available: https://inside-docupedia.bosch.com/confluence/aeos/core-assets/system-and-application/students-page

[4] Bob. (2011, Nov. 23). Hyundai Genesis [Online]. Available: https://automaniac.in/2012-hyundai-genesis

[5] K. Shin and Y. Yun, "A pre-crash discrimination system for an airbag deployment algorithm," *Proceedings of the 2010 American Control Conference*, 2010, pp. 6949-6954.

[6] Insurance Institute for Highway Safety. [Online]. Available: https://www.iihs.org/ratings/about-our-tests#frontal-crash-tests-1995-present

[7] Mike Linstromberg, Gerd Scholpp and Oliver Scherf, “Test and Simulation Tools in a Rollover Protection Development Process,” in *Siemens restraint Systems GmbH, ESV Conference, Washington,* 2005.

[8] Gabler, C. Hampton and John Hinch, "Evaluation of advanced air bag deployment algorithm performance using event data recorders," *in Annals of Advances in Automotive /Annual Scientific Conference in the Association for the Advancement of Automotive,* 2008, Vol. 52, pp. 175.

[9] M. E. Farmer and A. K. Jain, "Smart Automotive Airbags: Occupant Classification and Tracking," *in IEEE Transactions on Vehicular Technology*, vol. 56, no. 1, 2007, pp. 60-80.

[10] Pedram Aliabbasi, "Increasing Development Efficiency Using Virtual Prototyping in Automotive Domain," *in Chemnitz University of Technology, Faculty of Computer Science, Professorship of Computer Engineering*, 2018.

[11] Prasad Bhote, Owes Khan and Wolfram Hardt, "Latest trends in Continuous Integration for Highly Autonomous Driving," *in IBS International Symposium on Computer Science, Computer Engineering and Educational Technology*, 2019, pp. 49.

[12] M. Maniruzzaman, "Evaluation of Microcontroller Simulation for Transmission Control Units of Passenger Cars," *in Chemnitz University of Technology, Faculty of Computer Science, Professorship of Computer Engineering*, 2019.

[13] Nobert Englisch, Owes Khan, R. Mittag, F. Hänchen, Ariane Heller and Wolfram Hardt, “YellowCar: Automotive Multi-ECU Demonstrator Platform,” *in INFORMATIK*, 2017.

[14] Hasan Aljzaere, “Adaptive User Interface for Automotive Demonstrator,” *in Chemnitz University of Technology, Faculty of Computer Science, Professorship of Computer Engineering*, 2021.

[15] N. Englisch, R. Mittag, F. Hänchen, O. Khan, A. Masrur, and W. Hardt, “Efficiently Testing AUTOSAR Software Based on an Automatically Generated Knowledge Base,” *in the Proceedings of the 7th Conference on Simulation and Testing for Vehicle Technology*, May 2016, pp. 87–97.

[16] Stefan Förster, “A Formal Framework for Modeling Component Extension and Layers in Distributed Embedded Systems,” *in Chemnitz University of Technology, Faculty of Computer Science, Professorship of Computer Engineering*, January 2007.

[17] M. Springwald, R. Schmidt, and W. Hardt, “Design of a tool set for automated feedback on scientific writing,” *in IBS International Symposium on Computer Science, Computer Engineering and Educational Technology*, 2019, pp. 23.

[18] Kalkin Powale, “Automotive Powertrain Software Evaluation Tool,” *in Chemnitz University of Technology, Faculty of Computer Science, Professorship of Computer Engineering*, 2021.

[19] Bruce F. McNally, “Retrieving and Interpreting Data from Ford Powertrain Control Modules using the Bosch Crash Data Retrieval Tool,” *in CDR Conference on Software Engineering*, 2010.

[20] T. Karayiannis, J. Mades, T. Schneider, A. Windisch and W. Ecker, "Using XML for Representation and Visualization of Elaborated VHDL-AMS Models," *in Proceedings of VHDL International Users Forum Fall Workshop*, 2000, pp. 83-87.

[21] G. Goyal, K. Singh and K. R. Ramkumar, "A Detailed Analysis of Data Consistency Concepts in Data Exchange Formats (JSON & XML)," in International Conference on Computing, Communication and Automation (ICCCA), 2017, pp. 72-77.

[22] C. Stark, C. Medrano-Berumen and M. I. Akbaş, "Generation of Autonomous Vehicle Validation Scenarios Using Crash Data," *in IEEE Southeast Conference*, 2020, pp. 1-6.

[23] Ferrier, Brendon, Jim Lee, Alex Mbuli, and Daniel A. James. “Translational Applications of Wearable Sensors in Education: Implementation and Efficacy.” *in Sensors*, 2022, no. 4: 1675.

[24] S. Amann, S. Proksch, S. Nadi and M. Mezini, "A Study of Visual Studio Usage in Practice," *in IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, pp. 124-134.

[25] Johannes Lessmann and Frank Kelso and Wolfram Hardt, “Generic Integration Infrastructure for IP-based Design Processes and Tools with a Unified XML Format,” *in Integration - the VLSI journal*, September 2004, pp. 289 - 321.

[26] Garcia-Sanchez, P., Merelo, J.J., Laredo, “Evolving XSLT Stylesheets for Document Transformation,“ in Parallel Problem Solving from Nature – in Computer Science, 2008, vol 5199.

[27] K. P. Sreevishakh and S. P. Dhanure, "A Review Paper on Automotive Crash Prediction and Notification Technologies," *in International Conference on Computing Communication Control and Automation*, 2015, pp. 999-1002.

[28] C. Ruff, A. Eichberger and T. Jost, “Simulation of an Airbag Deployment in Out-of-Position Situation,” *in Vehicle System Dynamics for International Journal of Vehicle Mechanics and Mobility*, 2007, pp. 5-16.

[29] H. Yu, D. Zhang and N. Zhang, "Quality Traceability Technology for Automotive Airbag," *in Fifth International Conference on Intelligent Computation Technology and Automation*, 2012, pp. 269-271.

[30] S. Björnander, R. Land, P. Graydon, K. Lundqvist and P. Conmy, "A Method to Formally Evaluate Safety Case Evidences against a System Architecture Model," *in IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, 2012, pp. 337-342.

[31] C. Henard, M. Papadakis, M. Harman, Y. Jia and Y. Le Traon, "Comparing White-Box and BlackBox Test Prioritization,“ *in IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2016, pp. 523-534.