Maintenance & Repair Reporting:

Better Data and Advanced Analytics

Technomics, Inc.

Paul Hardin, Alexander LoRusso, Tyler Staffin

2/24/2020

ICEAA 2020 Paper

1225 South Clark Street, Suite 1500
Arlington VA, 22202
www.technomics.net

Abstract

The 1921-M/R (Maintenance & Repair Parts Data Report) is one component of the Department of Defense (DoD) system for collecting actual maintenance event and repair part data as part of the Cost and Software Data Reporting (CSDR) system. The CADE data repository serves as the primary source for contract cost, software, and technical data for many DoD resource analysis efforts, including but not limited to Lifecycle Cost Estimates (LCCEs). The -M/R will be used by DoD Component staff (i.e., program managers, systems engineers, cost estimators, and financial managers) to:

- Review and evaluate maintenance event, line-replaceable unit (LRU)/depot level reparable (DLR), repair part cost, and failure data
- Determine cost drivers and root cause
- Understand reasons for cost and availability performance
- Develop improved cost estimating techniques

The R Shiny package, used for the construction of interactive web applications via the R programming language, will be leveraged to analyze and demonstrate the value of M/R data to the cost community. Additionally, this paper will explore the mechanics and complex capabilities of the R Shiny framework in the context of advanced data analytics.

Maintenance & Repair Reporting:

Better Data and Advanced Analytics

PAUL HARDIN, ALEXANDER LORUSSO, TYLER STAFFIN

TECHNOMICS, INC.

Technomics

1225 South Clark Street, Suite 1500
Arlington VA, 22202
www.technomics.net

I. Introduction

The purpose of this paper is to inform the cost community of the 1921-M/R (Maintenance and Repair Parts) data reporting requirement and demonstrate its value through the application of advanced data analytics. The paper also provides an introduction to R and R Shiny, and how they were used to develop a dashboard reflecting the -M/R data.

Over the past few years, the Office of the Secretary of Defense (OSD) Cost Assessment and Program Evaluation (CAPE) has developed additional data reporting requirements to improve collection of sustainment cost data and related technical data from contractors with the intent of improved cost estimating capability. One of these areas of improvement includes the 1921-M/R (Maintenance and Repair Parts) Report.

Total sustainment phase cost can represent two-thirds or more of the total life-cycle cost of a major defense acquisition program (MDAP) and maintenance cost is a major component of total sustainment cost. Collection of M/R data for MDAPs reliant on contracted sustainment efforts is vital to improving cost management, cost reduction initiative investment and outcomes, and cost estimates of future programs.

The final version of the -M/R Data Item Description (DID) was completed recently and a number of programs are reporting actual M/R data. During the past year, a study was performed to review (i.e., verify and validate) the actual data, identify and correct any data issues, and use data analytics to confirm the value of the reported data.

During the study, the preliminary data analytics process was developed in Microsoft Excel to perform simple data verification and validation (V&V) checks. Later, it was later determined that a more standard, efficient, and capable approach would pay future dividends.

Various approaches were considered and led to development of a dashboard design using R, RStudio, and external packages. Although this effort is continuing, a significant portion of the dashboard and its functionality have been developed as described in the paper. The intent is for the M/R Data Dashboard to enable both the required data V&V and cost estimating applications in the future.

II. M/R Reporting

Description

Total sustainment phase cost can represent two-thirds or more of the total life-cycle cost of a major defense acquisition program (MDAP). The impetus for this paper, maintenance cost, is a major component of total sustainment cost. Collection of M/R data for MDAPs reliant on contracted sustainment efforts is vital to improving cost management, cost reduction initiative investment and outcomes, and cost estimates of future programs. The -M/R is the Department of Defense (DoD) system for collecting actual maintenance event and repair part data as part of the Cost, Software, and

Data Reporting (CSDR) System. OSD CAPE's Cost Assessment Data Enterprise (CADE) is DoD's only centralized source of contract cost, software, and technical data for use in many DoD resource analysis efforts, including cost database development, applied cost estimating, cost research, program reviews, Analysis of Alternatives (AoA), and Lifecycle Cost Estimates (LCCEs). The -M/R provides invaluable context to sustainment cost data in the form of technical data. This facilitates a better understanding of costs, leading to more accurate sustainment cost estimates.

The -M/R allows for collection of the following important data from defense contractors:

- Maintenance events
- Line replaceable unit (LRU)/depot level repairable (DLR)/repair parts cost
- Failure data

This data is equivalent to what is currently collected for organically repaired systems whose data is reported in the DoD Visibility and Management of Operating and Support Cost (VAMOSC) databases maintained by the Army, Navy and Air Force. Although some of this type of data has been collected in the past for contractor-supported programs (e.g., Joint Strike Fighter and Stryker Fighting Vehicle) as a Contract Data Requirements Lists (CDRLs) item, the objective of the -M/R Report is to institutionalize the requirement to ensure analysts have the same level of data detail for contractor-supported systems as organically repaired systems.

The -M/R will be used by program managers, systems engineers, cost estimators, and business/financial managers to:

- Review and evaluate maintenance event, LRU/DLR/repair part cost and failure data
- Identify demand and cost drivers
- Understand reasons for incurred cost and availability performance
- Develop improved cost estimating techniques

The -M/R consists of two separate reports shown below in Figure 1: -M/R Reports and Data Elements:

(1) **Maintenance Event Report** – collects information such as the specific system being repaired, location where the repair activity occurred, reason for failure, day failure was identified, and day repair activity was completed.

(2) **Repair Part Report** – identifies LRUs, DLRs and/or repair parts associated with each maintenance event.

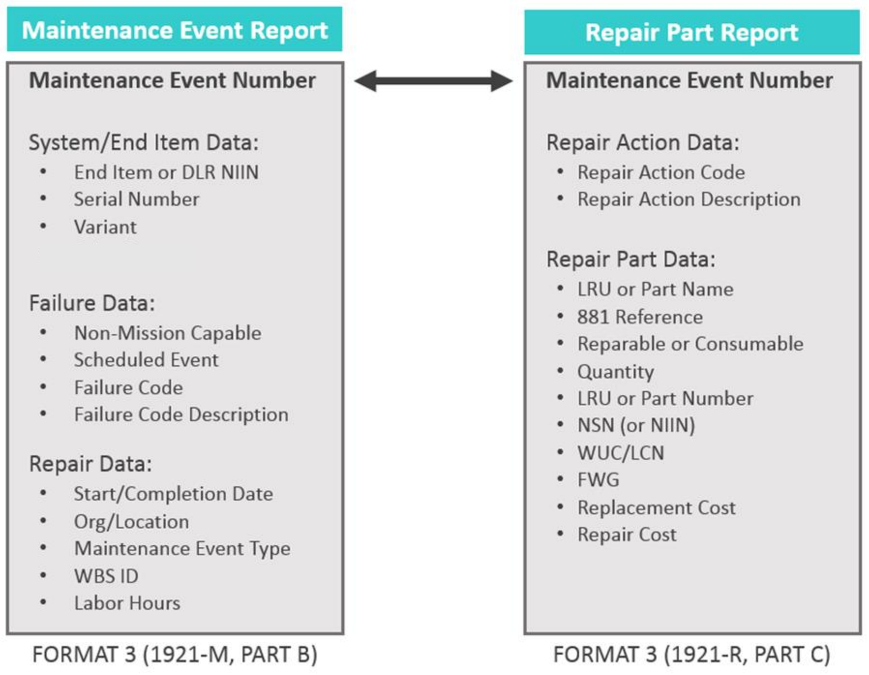


Figure 1: -M/R Reports and Data Elements

Initial Assessment

In 2019, a lessons-learned study was conducted to validate and assess actual reported data. This process resulted in a number of recommendations for improving aspects of the data. Additionally, initial data analytics were performed to confirm the value of M/R data.

The study results, including potential value-added applications of the data, were shared with the program office. Some of these applications are listed below.

- Assess and better understand top drivers by cost or demand, isolating drivers by Repair Part, Work Unit Code (WUC)/Functional Working Group (FWG), Maintenance Event Types, Repair Action Codes and Failure Codes
- Identify top reasons for failure (i.e., failure codes) for key Repair Parts
- Determine trends in repair actions, hours and cost per systems supported overtime and utilize information to improve cost estimates and decision-making
- Identify changes in cost and demand overtime in order to determine problem areas and develop potential improvements/solutions
- Capture critical maintenance management metrics such as:
  - Scheduled versus Unscheduled Activity
  - Hours per different maintenance event types
  - Days associated with events and/or repair parts
  - Current Replacement Cost of Repair Parts
  - Repair versus Replace Cost Ratios and Activity
  - Comparison of components to predicted reliabilities

- Failures occurring faster than, or not as quickly as expected

During the lessons-learned effort, the framework of the M/R data analytics process transitioned from Excel to R in order to more efficiently analyze various metrics, identify drivers, and dive deeper into problem areas of the program and its related detailed data. The following sections of this paper describe the use of R and the M/R Dashboard.

III. R Overview

There exist a variety of tools that provide an unprecedented ability to achieve insight from data. More recently, R has become a preferred data analytics solution in both the cost community and the greater data science world. The following content will highlight the main features of R, as well as describe how it provides deeper insight into M/R data when compared to alternative tools.

Introduction to R and R Shiny

Microsoft Excel tends to be the default platform used for computation and analysis within the cost community. However, when it comes to working with and analyzing large quantities of data, there are more suitable options such as R.

R is a free, functionally-oriented language and integrated platform developed for statistical computing, data manipulation, and advanced graphical display[1]. As an open-source framework, the capabilities of the software are expanding on a daily basis. Users across the globe leverage R for its fundamental and community-based features, and even contribute to the ongoing development of the language. There are nearly unlimited sources of documentation and code examples available on the web. More information on R and related content are detailed in Appendix B.

The R platform consists of a variety of specialized packages. Some of these packages, including the `base` package library, are essential for R to work. Other packages can be installed to serve a variety of other purposes and expand on the core features of the language.

Most users are familiar with the RStudio Integrated Development Environment (IDE), a software application that simplifies working with the language. In addition to creating the IDE, RStudio also develops a number of well-maintained packages. Built by RStudio, the `shiny` package provides an easy way to build dynamic, interactive dashboards in R. The package enables users to construct dashboards without requiring knowledge of full-stack web development. Shiny dashboards are fully customizable, and provide users with deeper insight into their data with a simple interface.

[1] https://www.r-project.org/about.html

R Shiny Framework

There are a few nuances of Shiny that must be understood prior to creating dashboards. Since the package facilitates web-application development, the structure may not be initially clear to first time R users. However, once the basics are mastered, the dashboard development process becomes relatively simple.

RStudio describes Shiny applications as being "a directory containing a user-interface definition, a server script, and any additional data, scripts, or other resources required to support the application[2]."

The user interface (UI) component is the portion of code where user-facing controls are established. This may consist of inputs (i.e. drop-down lists, buttons, sliders, fields, etc.), panels, sidebars, and other visual components. Most of what is seen on the screen is controlled by the defined structure of the UI.

The server function handles the back-end content of the dashboard. The server is the part of the application that enables R to accept given user input and generate a conditional response or output. This can include generating plots, updating drop-down lists, exporting files, running scripts, connecting to databases, and much more. The server is where all dynamic data manipulation occurs. Anything that can be performed in R can be executed on the server-side of a Shiny dashboard.

The distinction between UI code and server-side code is critical. All code must be placed within the appropriate component and ordered logically, or the dashboard will fail to render. Any error within the application script, whether a single letter or an entire block of code, will prevent a Shiny dashboard from launching. Thus, it is imperative that developers produce well-written code from the start. Since Shiny was employed as the framework for the M/R dashboard, it will be addressed in further detail later on.

IV. Developing the Dashboard

The M/R dashboard was built using the `shiny` package. The following section establishes requirements and explains why Shiny was selected as the framework for the M/R Dashboard.

M/R Dashboard Requirements

To understand why the M/R Dashboard was built in Shiny, it is helpful to first know the set dashboard requirements. The goal was to build a tool that highlights the importance of the M/R data, translating the data-rich reports into meaningful information to influence decisions. It was also

[2] https://rstudio.github.io/shiny/tutorial

essential to deliver a means for analysts to interact effectively with the data. In the design stage of the dashboard, the following requirements and features were outlined:

(1) Navigate from high-level to low-level information
(2) Visuals should have extensive customization
(3) Visuals render dynamically and show varying levels of detail based on user inputs
(4) User input functions are easy to use
(5) All visuals must be interactive
(6) Dashboard must be dynamic without sacrificing speed

Why Use R Shiny for M/R

"Shiny allows you to take your work in R and expose it via a web browser so that anyone can use it. Shiny makes you look awesome by making it easy to produce polished web apps with a minimum amount of pain[3]." – Hadley Wickham, Chief Scientist at RStudio

All of the requirements from the previous section were able to be satisfied within the context of a Shiny dashboard. These six requirements are expanded upon in more detail below:

(1) Navigate from high-level to low-level information

A key component of the dashboard was to allow the user to drill down and see the data at the lowest level. The user has the ability to explore the data at higher levels and then, based on the knowledge/insights gained from that information, query the dataset to identify cost drivers. In short, the intent was to allow the analyst flexibility by not limiting the ability to see differing levels of data detail. Many dashboards only provide views that inspire an analyst to go back and search the dataset for more information. This dashboard needed to provide a fast and simple way to search the dataset within the dashboard environment. Shiny provides a framework for showing high-level and low-level views in the same place. It becomes a one-stop shop for the entire analytical experience.

(2) Visuals should have near-limitless customization

One of the most effective ways for people to interpret data is through visualization. Every aspect of a visual can be tailored to the creator's liking. While there are certainly ways to customize graphs and charts in Excel, there is a point of limitation—and that point may be quickly reached. The font of the data labels is perfect, the color scheme matches the company's branding, but the arrowhead desired is not among the arrowheads available. In Shiny, that point is much harder to reach. The

[3] http://www.mastering-shiny.org

code behind all visuals is designed to allow customization to extreme lengths. This tends to be rather simple as well, given that the functions provide parameters to change all the aspects of a visual.

(3) Visuals render dynamically and show varying levels of detail based on user inputs

The dashboard was required to contain graphs with the ability to show varying levels of detail based on user inputs. The visuals were designed to start at a high level, and go lower as more inputs are specified. In an application such as Excel, each added layer of filtering is likely to complicate the set-up, and worse, hinder performance. R is highly scalable, ensuring graphics are rendered dynamically and efficiently, regardless of the size of the underlying dataset.

(4) User input functions are easy to use

User inputs functions should be intuitive and easy to use, preferably with buttons and drop-down menus rather than manual text entry. Shiny has pre-coded functions that allow for a variety of different inputs, all of which are clean and work similarly in structure. It is simply a matter of choosing which one. In addition, combinations of different of input methods can be used to dynamically filter a single visual or data table.

(5) All visuals must be interactive

Given R is designed to increase usability of data, it is natural that R supports cutting-edge data visualization. Powerful graphics deliver more insight into the data than standard charts provide. It is easy to zoom in and out, change the graph region, reveal hover labels over each point, and export visuals. All of these features happen instantaneously, and do not impact rendering time. The ability to interact with visuals as analysis is performed greatly improves the data analysis process. It transforms the analytical experience from viewing static pictures of graphs in a slideshow to interactively engaging with the data.

(6) Dashboard must be dynamic without sacrificing speed

Most importantly, the dashboard needed to be as dynamic as possible while still maintaining speed. Each input triggers intensive manipulation of a dataset that is then used to render interactive plots. These updates must be instantaneous. In order to limit stress and reduce the amount of time necessary to perform these actions, the dashboard required great computing power. The environment should not be the limiting factor in how quickly analyses can be performed, and this is Shiny's bread-and-butter. The entire framework is designed to allow for everything to be dynamic. Complex dynamic features can be added without significantly sacrificing speed. In a Shiny dashboard, there is greater ability to strategize when calculations occur and design it in such a way that will result in ideal performance catered to the user's preference.

V. The M/R Dashboard

The 'real' M/R Dashboard includes actual data collected for DoD weapon systems. For the purpose of demonstrating the tool and describing the functionality in this paper, fictitious data was employed.

Dataset Explanation

A fictitious dataset containing ground vehicle M/R data was generated to demonstrate use of the M/R Dashboard. For instance, this dataset could provide insight to a program manager into cost drivers for an existing ground vehicle program. A program manager may be interested in comparing maintenance events and cost drivers between automatic and manual variants of a certain vehicle type. The dashboard was constructed to work with any dataset matching the same format and structure, thus it was effortless to replace the actual program dataset with the fictitious dataset. The following is a detailed overview of the categorical variables from the M/R report that were used in the dashboard:

Variant – each vehicle is categorized by an overarching variant type. The two variants in this dataset are based on different transmission types: manual and automatic.

Functional Working Group (FWG) – each part of the vehicle is bucketed into a higher-level category containing parts serving a similar function. These include systems such as Electrical, Brake, Exhaust, Body, etc.

Part Name – the part name is the lowest level of data in the dataset. Examples of part names included are the Battery, Pedal, Fender, Exhaust Manifold, etc. Each of these parts map to one of the discrete FWGs.

Event Type – describes the type of maintenance event. The categories of maintenance events include Routine, Modification, Accident, Inspection, Wear, and Malfunction.

Failure Code – a three-digit number that corresponds to the type of part failure. This is the code that the vehicle's diagnostics data reveals when examined.

Schedule Status – tags the maintenance event as either a scheduled or unscheduled event.

Additionally, there are a number of numeric variables reported in the dataset that were used in the dashboard. These variables include the maintenance period start date, repair part quantity, labor hours, replacement cost, repair cost, and total cost. An additional monthly-phased dataset was merged with this one to provide insight into the total number of units on the road, number of trips taken, total amount of driving time, and average hours per trip.

The `readr` package was used to import the data from CSV format into the R environment, and `dplyr` and `tidyr` were employed to restructure the data and calculate additional metrics for the dashboard. The process of importing and manipulating the data in R occurs in just a few seconds. After this, the prepared data is pushed through the Shiny framework, and the dashboard is displayed nearly instantaneously.

Dashboard Walkthrough

With a relatively small amount of code, R can successfully move raw data into advanced dashboard displays, providing decision-makers with instant insight into their data. The following figures (larger views of which can be seen in Appendix A) are the main views from the current iteration of the M/R Dashboard:

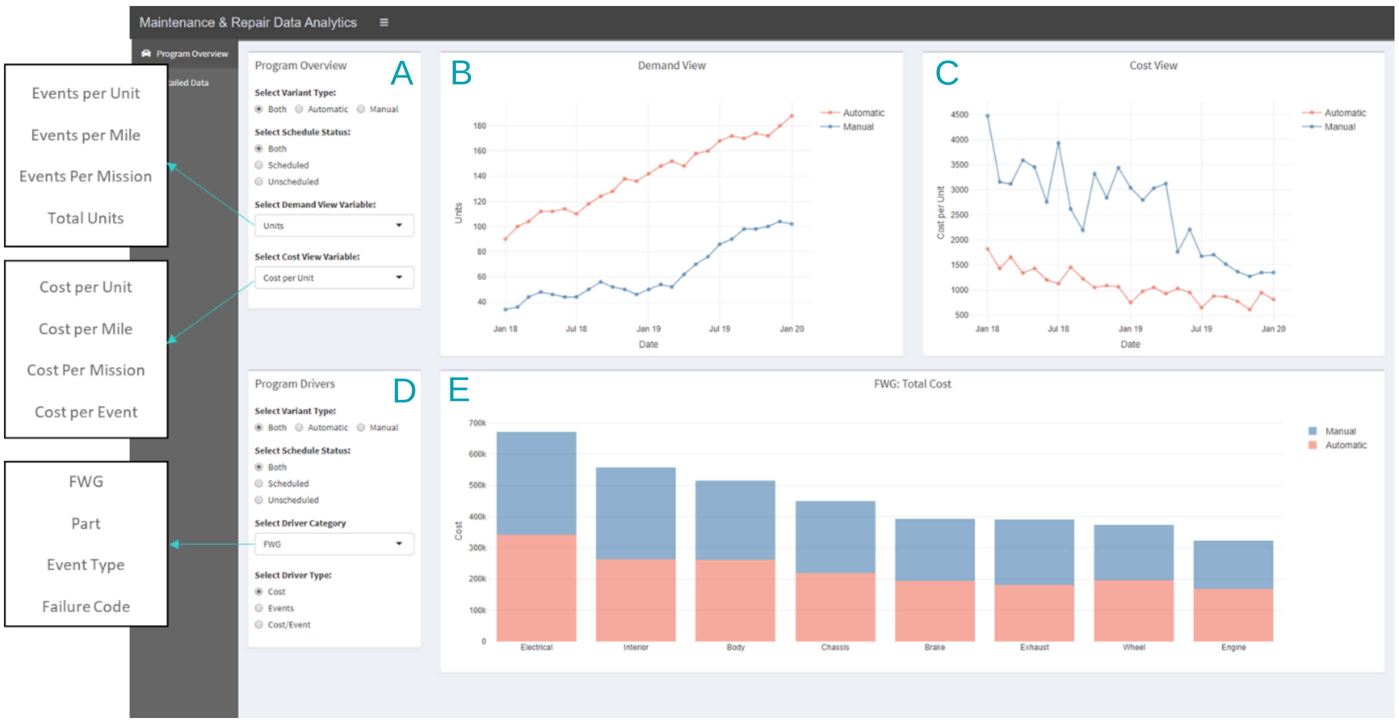


Figure 2: Dashboard Program Overview

(A larger view can be seen in Appendix A.)

A – This input box dynamically controls the output of plots B and C. Users have the option to filter the data to the different variant types and schedule statuses, and can select specific variables for the corresponding plots.

B – This plot visualizes the Demand View of the data on a monthly basis. Users select from the drop-down list in input box A to toggle between variables, including Events per Unit, Events per Mile, Events per Mission, and Total Units.

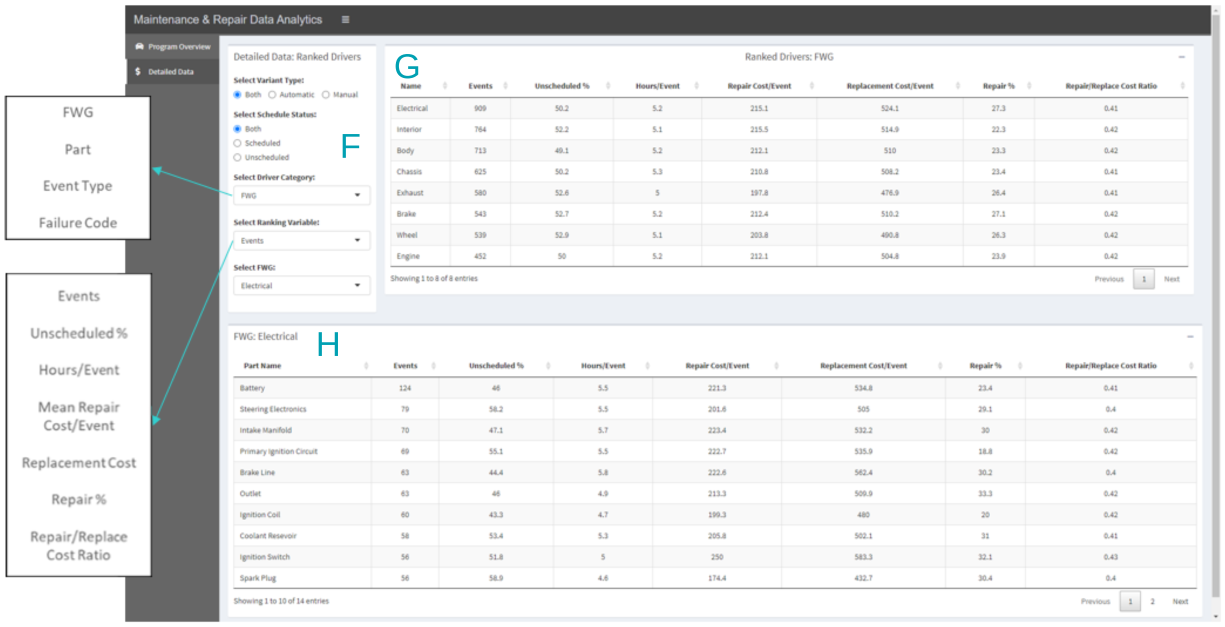
C – This plot visualizes the Cost View of the data on a monthly basis. Users select from the drop-down list in input box A to toggle between variables, including Cost per Unit, Cost per Mile, Cost per Mission, and Cost per Event.

D – This input box dynamically controls the Program Driver output in plot E. Users have the option to filter the data to the different variant types and schedule statuses, as well as choose the driver category and driver type. Driver categories include FWG, Part, Event Type, and Failure Code. Driver types include Cost, Events, and Cost/Event.

E – This plot visualizes the Program Driver View of the data, and its output depends on the user selections for input box D.

For example, the current input settings for A (Figure 2) result in a view that compares unit-level and total cost drivers across both variants (for both schedule statuses). The total number of units increases within the date range, with significantly more Automatic ground vehicles deployed than their Manual counterparts (B). Over time, the total cost per unit decreases for both variants, and the total cost per unit of the Manual variant approaches that of the Automatic variant. The FWG driver category and total cost driver type are selected (in D), thus the bottom bar graph (E) reveals the largest cost drivers by FWG (in this case, Electrical).

Figure 3 demonstrates the process of examining ranked drivers by FWG, as well as the ability to deep-dive into the drivers by part for a given FWG. FWG is selected from the driver category drop-down (F), revealing ranked drivers by FWG (in G). Since FWG is selected as the driver category, an additional drop-down selection list appears, containing the various FWGs. Electrical is selected as it is the largest program cost driver, and an additional lower-level table (H) is revealed. H lists the largest part-level drivers for the Electrical FWG and their corresponding M/R metrics, sorted by total number of maintenance events (based on the drop-down input in F). With ease, the dashboard inputs can be modified to generate views pertaining to other metrics and variables from the dataset.



Detailed Data: Ranked Drivers

Select Variant Type: [✓ Both] Automatic Manual

Select Schedule Status: [✓ Both] Scheduled Unscheduled

Select Driver Category: FWG

Select Ranking Variable: Events

Select FWG: Electrical

Ranked Drivers: FWG

Name	Events	Unscheduled %	Hours/Event	Repair Cost/Event	Replacement Cost/Event	Repair %	Repair/Replace Cost Ratio
Electrical	909	50.2	5.2	215.1	524.1	27.3	0.41
Interior	764	52.2	5.1	215.5	514.9	22.3	0.42
Body	713	49.1	5.2	212.1	510	23.3	0.42
Chassis	625	50.2	5.3	210.8	508.2	23.4	0.41
Exhaust	580	52.6	5	197.8	476.9	26.4	0.41
Brake	543	52.7	5.2	212.4	510.2	27.1	0.42
Wheel	539	52.9	5.1	203.8	490.8	26.3	0.42
Engine	452	50	5.2	212.1	504.8	23.9	0.42

Showing 1 to 8 of 8 entries

FWG: Electrical

Part Name	Events	Unscheduled %	Hours/Event	Repair Cost/Event	Replacement Cost/Event	Repair %	Repair/Replace Cost Ratio
Battery	124	46	5.5	221.3	534.8	23.4	0.41
Steering Electronics	79	58.2	5.5	201.6	505	29.1	0.4
Intake Manifold	70	47.1	5.7	223.4	532.2	30	0.42
Primary Ignition Circuit	69	55.1	5.5	222.7	535.9	18.8	0.42
Brake Line	63	44.4	5.8	222.6	562.4	30.2	0.4
Outlet	63	46	4.9	213.3	509.9	33.3	0.42
Ignition Coil	60	43.3	4.7	199.3	480	20	0.42
Coolant Resevoir	58	53.4	5.3	205.8	502.1	31	0.41
Ignition Switch	56	51.8	5	250	583.3	32.1	0.43
Spark Plug	56	58.9	4.6	174.4	432.7	30.4	0.4

Showing 1 to 10 of 14 entries

Figure 3: Dashboard Detailed Data View

(A larger view can be seen in Appendix A.)

F – This input box dynamically controls the Ranked Driver Table (G). Users have the option to filter the data to the different variant types and schedule statuses, as well as choose the driver category. Users also have the ability to choose the ranking variable from the table.

G – This table enables users to view the data for the various driver categories. Such variables include number of maintenance events, percentage of unscheduled maintenance events, total hours per maintenance event, average repair cost per maintenance event, total replacement cost, percent of repairs (the percentage of maintenance events were specifically repairs), and the cost ratio of repair events to replacement events (the amount of cost to repair relative to cost to replace).

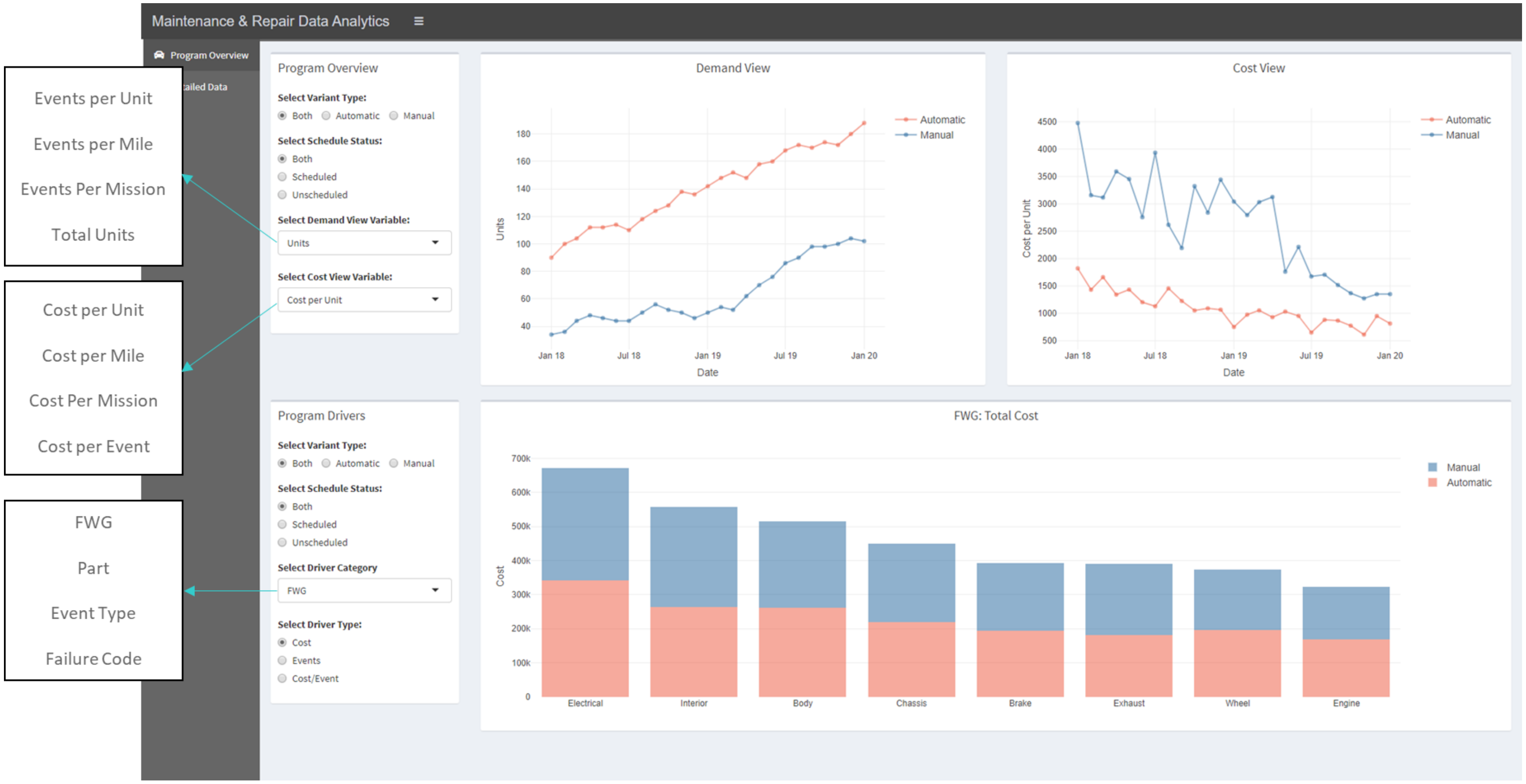
With this dashboard, users achieve rapid insight into a number of elements in their dataset. This includes identifying demand and cost drivers, identifying key reasons for part failure, understanding time-phased trends in data, analyzing various demand and cost metrics, and tracking overall performance over time. Depending on user needs for a particular program, features can be added to or removed from this dashboard with only a few simple lines of code. In the future, the dashboard will be updated to include a deep-dive capability for drivers by WUC (Work Unit Code)/LCN (Logistics Control Number) and MIL-STD-881 WBS (Work Breakdown Structure).

VI. Conclusion

The sustainment phase can represent two-thirds or more of the total life-cycle cost of a major defense acquisition program (MDAP). The impetus for this paper, maintenance cost, is a major component of total sustainment cost. Collection of M/R data for MDAPs reliant on contracted sustainment efforts is vital to improving cost management, cost reduction initiative investment and outcomes, and cost estimates of future programs. The technical data being collected as part of DoD's CSDR requirement imposed on defense contractors provide context to and offer the potential to explain cost. Understanding maintenance-related cost drivers well enough to pose the 'right' questions to peel the onion effectively and identify root causes is critical to estimating, managing, and reducing costs.

Collecting improved contractor maintenance and repair data and applying advanced data analytics capabilities is essential for improving our cost community capability. This paper has provided background regarding invaluable M/R data being collected today, and details regarding R and its use in developing an innovative M/R dashboard that will enable program managers, systems engineers, cost estimators and business/financial managers to exploit the M/R data.

Appendix A



Maintenance & Repair Data Analytics

Program Overview

Select Variant Type: [✓ Both] Automatic Manual

Select Schedule Status: [✓ Both] Scheduled Unscheduled

Select Demand View Variable: Units

Select Cost View Variable: Cost per Unit

Program Drivers

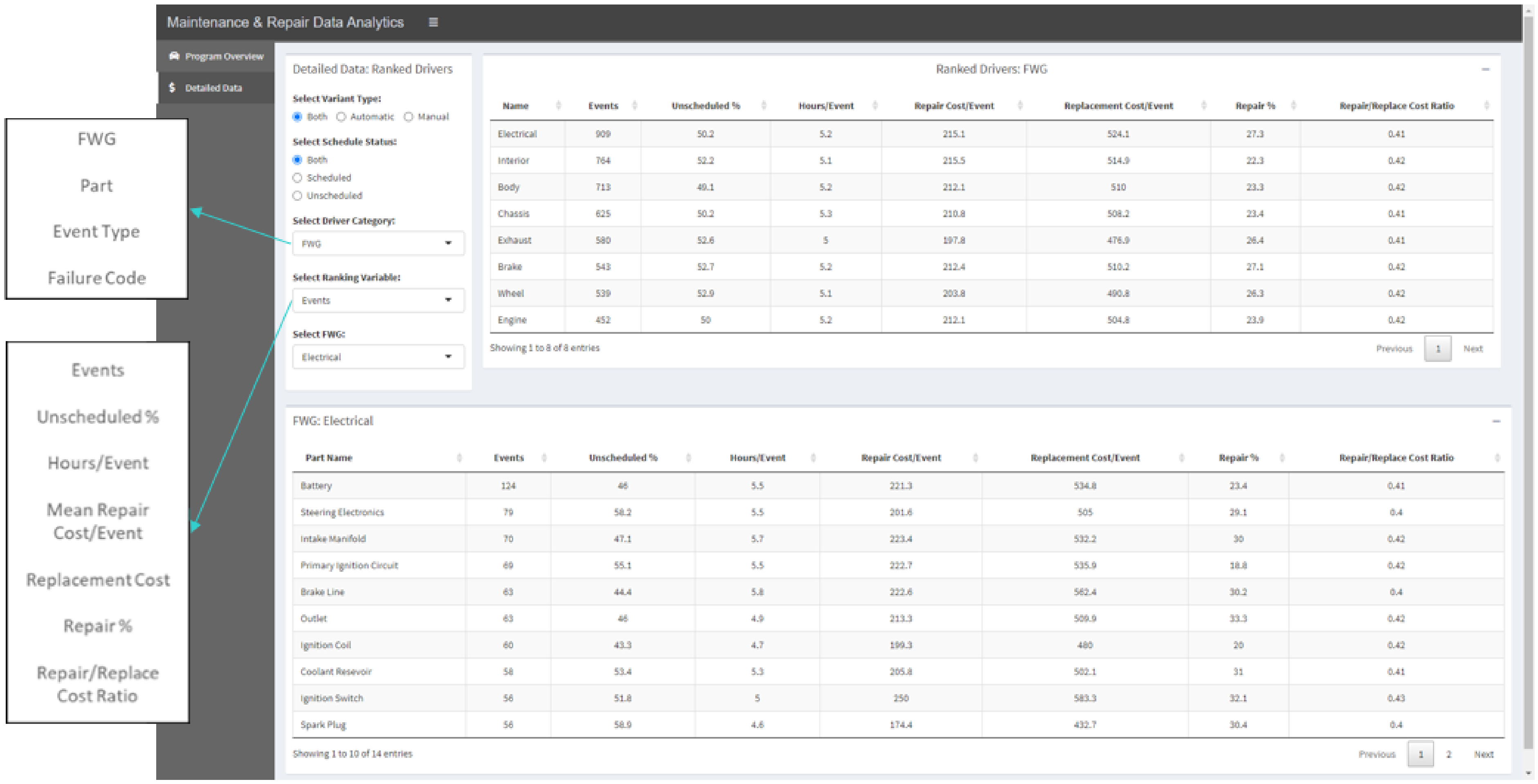
Select Variant Type: [✓ Both] Automatic Manual

Select Schedule Status: [✓ Both] Scheduled Unscheduled

Select Driver Category: FWG

Select Driver Type: [✓ Cost] Events Cost/Event

Figure 2: Dashboard Program Overview



Detailed Data: Ranked Drivers

Select Variant Type: [✓ Both] Automatic Manual

Select Schedule Status: [✓ Both] Scheduled Unscheduled

Select Driver Category: FWG

Select Ranking Variable: Events

Select FWG: Electrical

Ranked Drivers: FWG

Name	Events	Unscheduled %	Hours/Event	Repair Cost/Event	Replacement Cost/Event	Repair %	Repair/Replace Cost Ratio
Electrical	909	50.2	5.2	215.1	524.1	27.3	0.41
Interior	764	52.2	5.1	215.5	514.9	22.3	0.42
Body	713	49.1	5.2	212.1	510	23.3	0.42
Chassis	625	50.2	5.3	210.8	508.2	23.4	0.41
Exhaust	580	52.6	5	197.8	476.9	26.4	0.41
Brake	543	52.7	5.2	212.4	510.2	27.1	0.42
Wheel	539	52.9	5.1	203.8	490.8	26.3	0.42
Engine	452	50	5.2	212.1	504.8	23.9	0.42

Showing 1 to 8 of 8 entries

FWG: Electrical

Part Name	Events	Unscheduled %	Hours/Event	Repair Cost/Event	Replacement Cost/Event	Repair %	Repair/Replace Cost Ratio
Battery	124	46	5.5	221.3	534.8	23.4	0.41
Steering Electronics	79	58.2	5.5	201.6	505	29.1	0.4
Intake Manifold	70	47.1	5.7	223.4	532.2	30	0.42
Primary Ignition Circuit	69	55.1	5.5	222.7	535.9	18.8	0.42
Brake Line	63	44.4	5.8	222.6	562.4	30.2	0.4
Outlet	63	46	4.9	213.3	509.9	33.3	0.42
Ignition Coil	60	43.3	4.7	199.3	480	20	0.42
Coolant Resevoir	58	53.4	5.3	205.8	502.1	31	0.41
Ignition Switch	56	51.8	5	250	583.3	32.1	0.43
Spark Plug	56	58.9	4.6	174.4	432.7	30.4	0.4

Showing 1 to 10 of 14 entries

Figure 3: Dashboard Detailed Data View

Appendix B

R

Since R does not have a spreadsheet-based user interface as Excel does, everything must be coded manually. The transition from spreadsheets to code may be initially challenging, but can open the door to significantly more capability when working with data. Listed are some of the fundamental capabilities of the R platform that make it a highly viable option:

- **Traceability** – code is traceable by nature. All calculations are visible.
- **Speed** – the only limiting factors for speed are code inefficiencies, core system limitations, and size of any integrated data sets.
- **Repeatability** – the scripting environment is highly conducive to task and process automation. R integrates well with other software and external databases.
- **Complexity** – R can handle tasks varying from basic addition to machine learning on big data.
- **Price** – R is free for all major operating systems, while similar platforms are not.

As a result, the popularity of R has increased within the cost community and in the greater data science field. Technologies such as R continue to challenge the way people have traditionally solved quantitative problems, and enable government and industry professionals to handle data in a more efficient and suitable manner.

The process of working with data in R is inherently different from that of spreadsheet-based platforms. The main data structures in R consist of vectors, matrices, data frames, and lists, each of which is addressed below.

- **Vectors** – data structures that contain elements of the same type. Types include character, numeric, logical, complex, factor, and others.
- **Matrices** – collections of vectors in an array that contain elements of the same data type.
- **Data frames** – collections of vectors in an array that may not necessarily contain elements of the same data type.
- **Lists** – data structures that hold other data structures. Each data structure can vary within a list (i.e. a list containing a data frame and a vector)

Data frames are the most common data structures for storing, analyzing, and visualizing data in R. Each of the columns of a data frame are individual vectors (with the same number of elements), thus data tends to be manipulated on a columnar basis. Subsets of data frames can be easily generated, additional columns can be added, other data frames can be joined, and individual elements can be edited. Data frames are conceptually similar to tables in Excel.

RStudio

Once installed, R can be accessed through its basic Graphical User Interface (GUI) or through the system terminal. Most users are familiar with the RStudio Integrated Development Environment (IDE), a software application that simplifies working with the language. The main features of

RStudio include a basic text editor, console, plot viewer, package manager, and a documentation panel, among many others.

Another great feature that the application provides is the ability to selectively run portions of code, instead of requiring entire scripts be executed in chronological order. RStudio also makes it easy to connect to databases, run version control, navigate file directories, import datasets, and locate package documentation. Although RStudio is not a requirement to use R, it dramatically improves the user experience.

R Packages

The R platform consists of a variety of specialized packages. Some of these packages, including the `base` package library, are essential for R to work. Other packages can be installed to serve a variety of other purposes. Simply put, packages are files that contain various functions and datasets, and assist users with certain tasks or analysis methods. While most tasks in R can be tackled using only `base`, there are a number of widely recognized packages that enhance the language.

There are limitations of `base` R that can be handled via external packages. For example, `openxlsx` is a package for importing and exporting Excel data. This capability is not included in `base` R, so `openxlsx` and similar packages were developed to provide a simple way of interacting with Excel data. As R is an open-source language, anyone can contribute to the development of new and existing packages. There are over 15,000 publically accessible packages hosted on the Comprehensive R Archive Network (CRAN) to date, many of which are monitored and improved by developers across the globe.

In addition to building the IDE, the RStudio team has supported development of many commonly used packages, including the `tidyverse` family, `shiny`, and `shinydashboard`. These packages are highlighted in this paper, along with the `plotly` interactive visualization library. The `tidyverse` is a collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures[4]. Table 1 includes high-level overviews of some popular used packages. Packages from the `tidyverse` are denoted with an asterisk.

[4] http://www.tidyverse.org

Table 1: Popular Packages

dplyr*	Collection of functions used for improved data manipulation
ggplot2*	Library used for the development of simple to complex graphics
tidyr*	Helps restructure data into more standard formats
readr*	Promotes faster data importing and parsing
purr*	Provides tools for efficient function mapping to data structures
stringr*	Assists in processing and manipulation of text
shiny	Enables interactive dashboard development
shinydashboard	Simplifies the development of a dashboard user interface (UI) for shiny
plotly	JavaScript based library built for the creation of interactive graphics