## 1.0 Introduction

The Builder Connect Application (Henceforth referred to simply as the "Application") is a product that Kenna provides to its clients at BASF Construction North America. It is a comprehensive suite of tools and solutions that enable BASF marketing professionals to make better-informed decisions and to better engage with their customers. Among its many features, the Application derives much value from its Reporting Utilities. Easily accessible via the top navigation bar, the "Reports" sections of the Application shows various reports generated from the data the Application has collected from its users.

Because the Application accommodates for different classes of users, the Reports section likewise will also show different reports depending on the specific roles and responsibilities of the current user. Here, the author would like to highlight the Management Reports. Accessed via the "Management Reports" tab in the sliding side navigation bar (Figure 1.0), these reports are an indispensable source of information for users in an organizational role.

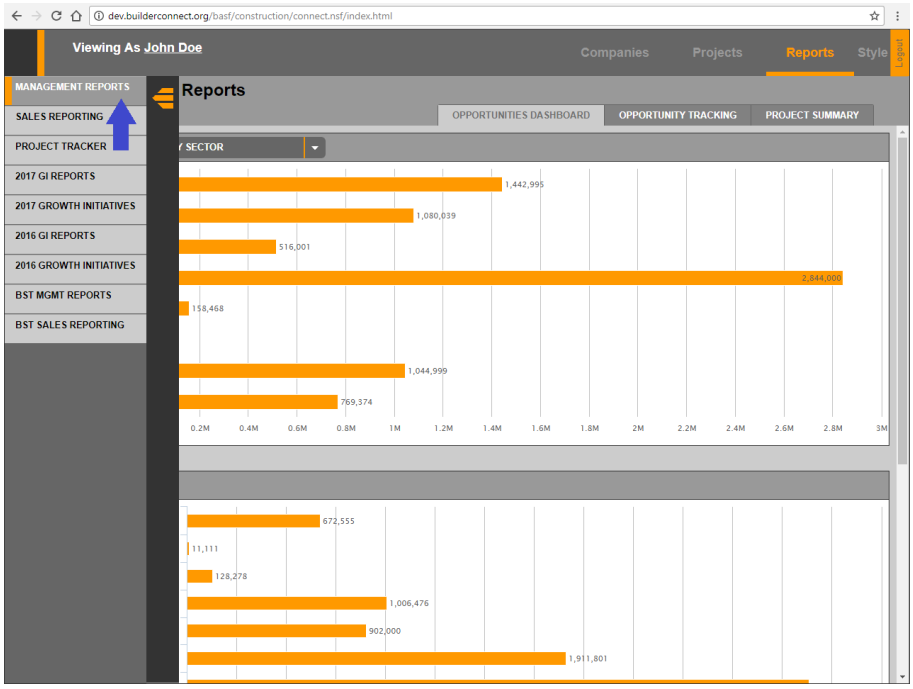


**Figure 1.0: Navigating to the Management Reports**

A key component of the Management Reports is the filters panel (Figure 1.1). This panel provides a number of controls which management users can use in order to fine-tune the dataset the reports draw from. This unassuming feature is a crucial component of the Management Reports because it provides the user with greater depth into their data, giving them multiple perspectives and, therefore, allowing them to optimize their future strategy. These filters are the focus of discussions in this report.

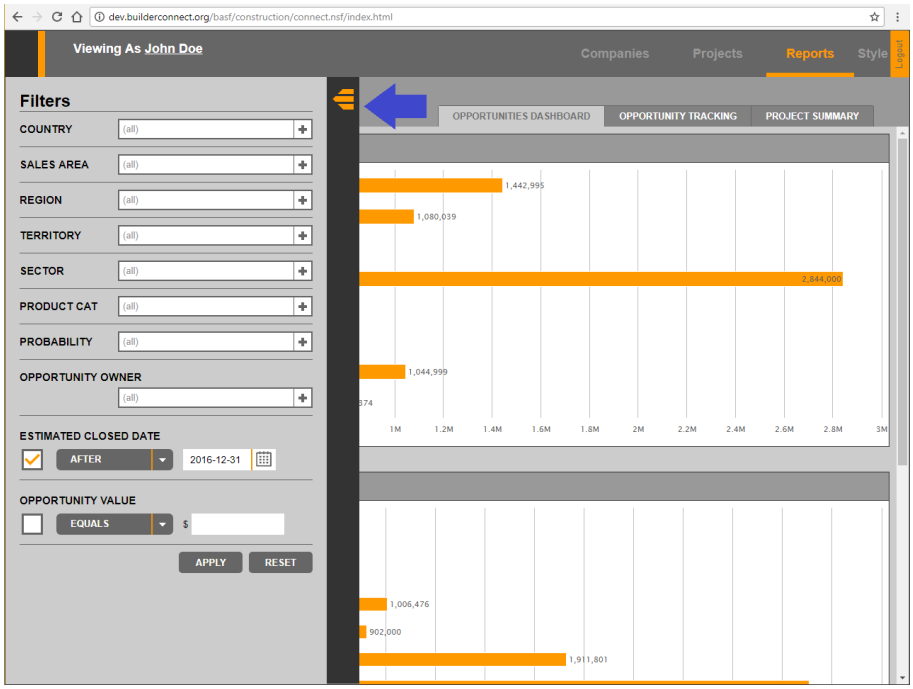


**Figure 1.1: The Management Report Filters**

When this author was tasked with adding a number of newer controls to the Management Report Filters, he had to examine the existing infrastructure to understand how the Filters worked. In doing so, the author discovered several points of weakness in the existing framework which created unnecessary complications and made future maintenance and expansion difficult. If steps are not taken to correct these shortcomings, then the development of not just the filters, but the Reports sections as a whole, is likely to be hindered. Thus, the purpose of this report is to

provide the first step in the process of improvement by describing and analyzing the deficiencies in the existing infrastructure.

## 2.0 – Analysis

### 2.1 – The Existing Infrastructure

For the aforementioned purpose, the author now finds it helpful to provide an overview of the old system.

In brief, the Management Report Filters work as follows:

1. When the Application first starts, a function called `init_reports()` is called.
2. This function creates a *closure*, within which a variable, `mgmt_report_filters`, is defined. This variable keeps track of the state of the controls in the filters panel.
3. When the user interacts with the filters panel, the selections and inputs they give are, for the most part, registered and stored in `mgmt_report_filters`.
4. When the user clicks on the "Apply" button in the filters panel, the contents of `mgmt_report_filters` is copied over to a "mirror" Object in the *closure* of the `initReport()` function.
5. This mirror Object is *stringified* and inserted into the header of various XMLHttpRequests (Performed via `$.ajax()`), thus sending the filter information to the backend and ensuring that the response returned is filtered.

Having outlined the system, the following sections will dive into the system in greater depth, with an emphasis on highlighting its weaknesses. Two things, however, must be noted before continuing. First, without exception, all subsequent code to be shown in this report comes from

the file reports.js. Second, in all code shown, the appearance of the sequence of characters /* ... */ is to denote code omitted for the purpose of keeping this report short and concise.

### 2.2 – An Inconsistent System

The first issue in the implementation of the Management Report Filters is the many inconsistencies it has both within itself and with other parts of the application. Here, the most concerning of these inconsistencies will be addressed.

To begin, the author would like to draw the reader's attention to another filters panel found within the Application. The Project List Filters (Figure 2.0) provides several filter controls which the user can use in order to find the projects they need.

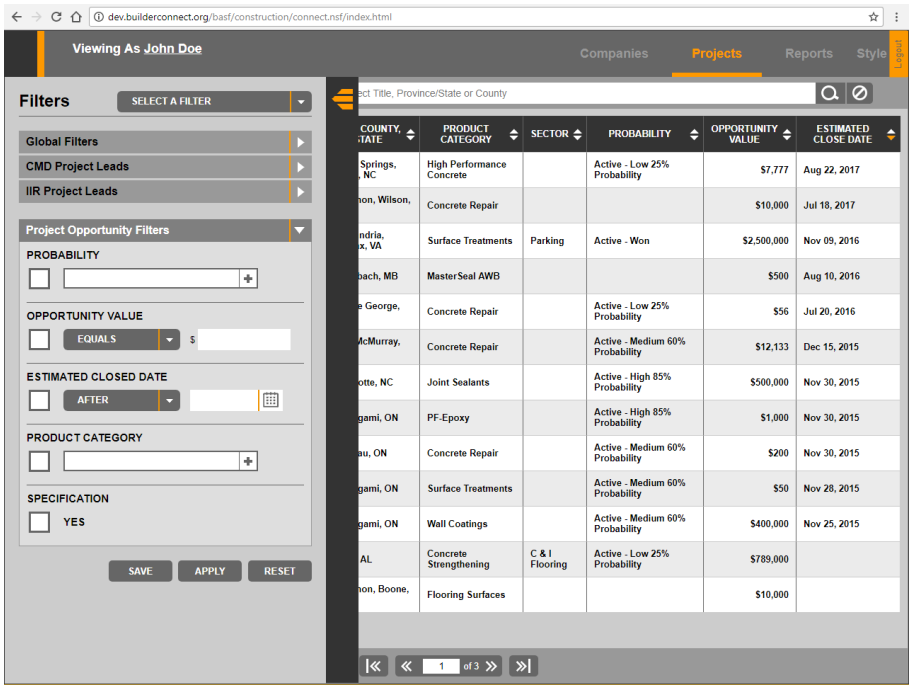


**Figure 2.0: The Project List Filters**

Several filter controls between the two filter panels are very much similar, if not exactly the same. Take, for example, the "Probability" filter control. In both filter panels, a click on the little white box with the "plus" symbol will open an overlay. The user can then make their selection in the overlay, click "Save," and see their selection show up in the box (Figures 2.1 to 2.3).

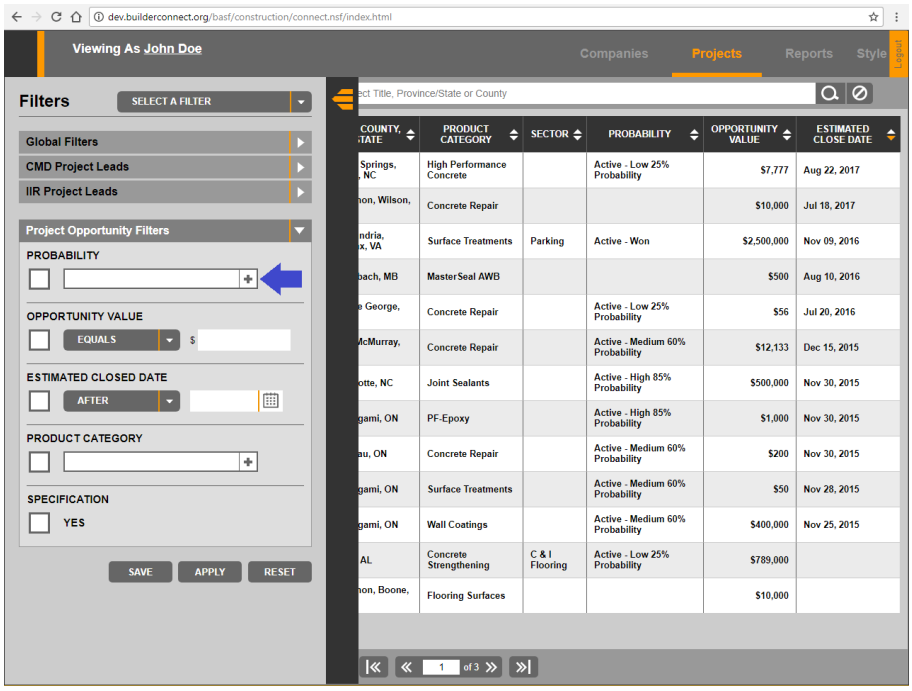


**Figure 2.1: A click on the white box opens an overlay window**

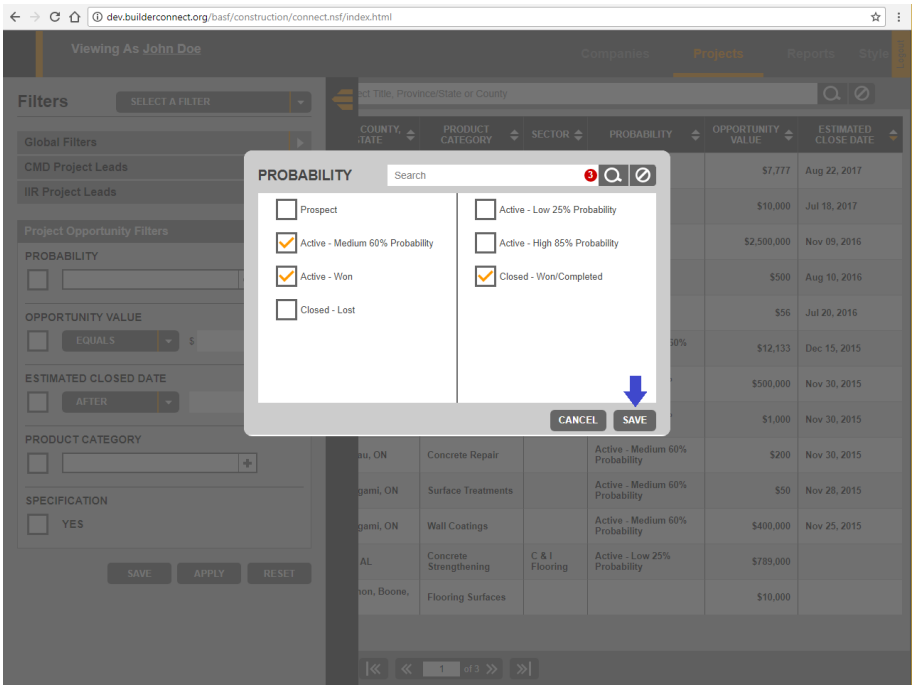


**Figure 2.2: The user can make their selection and click “Save”**

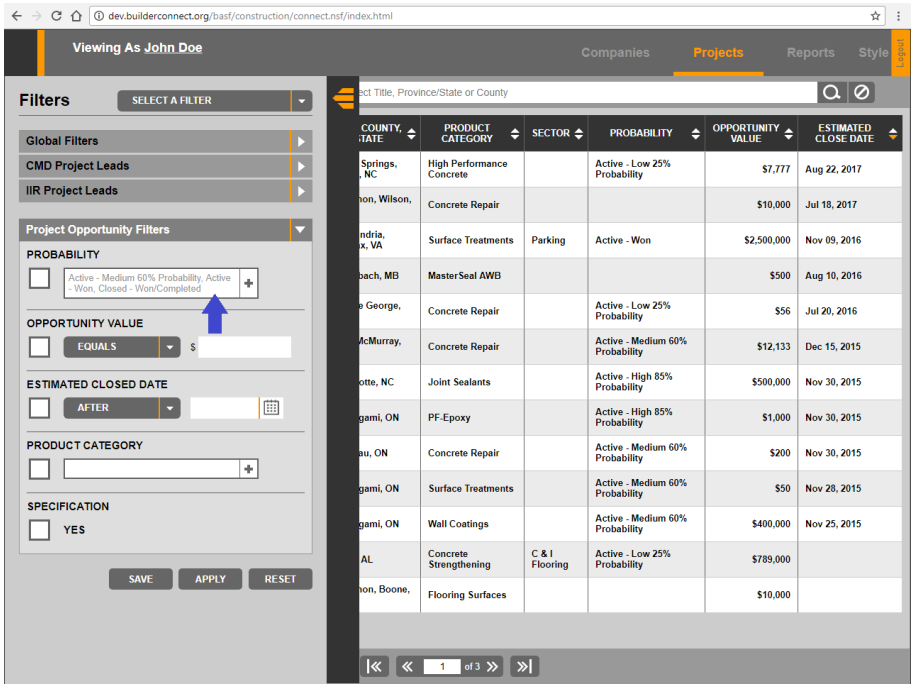


**Figure 2.3: The user’s selection will be displayed in the box**

As a result of their similarities in behaviour, one would expect that these two filters controls also share similar implementations. This, however, is not the case. The "Probability" filter in Project List Filters is implemented through what the author would like to call as the `$.list_filter()` strategy. The details of this strategy is too lengthy to be described here, but it is roughly as follows:

1. Choose a DOM element to be used as a list filter, initialize it by calling `$(selector).list_filter()`, where `selector` is a selector string that unique identifies the element.
2. As the user interacts with the element (e.g., Clicking on it to see an selection overlay), register the inputs given by the user and save them by calling `$(selector).list_filter('set_values', input)`, where `input` can be any value appropriate for representing the user's inputs.
3. When the filter information is needed (e.g., To be inserted into the URL of an XMLHttpRequest), get it by calling `$(selector).list_filter('get_values')`.

This strategy is widely adopted across the application for various filter controls in numerous different filters. But, for reasons unknown to the author, it is not used at all in the implementation of the Management Report Filters. A search of the term "list_filter" within `reports.js` yields no results.

Inconsistencies such as this scatter themselves across the implementation of the Management Report Filters and present challenges for future development as they force to developer to have to deal with multiple systems.

At this stage, one may be wondering how the "Probability" filter control and others like it are implemented in the Management Report Filters. Diving into this question reveals the next shortcoming of the Filters the author wishes to highlight.

### 2.3 – A Lack of Modularity

The DOM element used for the "Probability" filter control within the Management Report Filters has the attribute `data-role` set to the value `report-selection-overlay`. Figure 2.4, which is an excerpt from the `init_report()` function in `reports.js`, shows the attaching of a "click" event handler to this element.

```
// Note: body is a variable which holds document.body
body.on('click',
  '[data-role="report-selection-overlay"]'
  + ',[data-role="mgmt-report-selection-overlay"]'
  + ',[data-role="prjtrack-report-selection-overlay"]',
  function(e) {
    var $field = $(this);
    /* ... */
    var data_role = $field.attr('data-role');
    /* ... */
    if (data_role == "mgmt-report-selection-overlay") {
      /* ... */
    } else if (data_role == "prjtrack-report-selection-overlay") {
      /* ... */
    } else {
      /* ... */
    }

    getSelectionOverlayData(value_type, selected_filters, function(response) {
      /* ... */
      if (data_role == "mgmt-report-selection-overlay") {
        /* ... */
      } else if (data_role == "prjtrack-report-selection-overlay") {
        /* ... */
      } else {
        /* ... */
      }
      /* ... */
    });
  });
```

**Figure 2.4: The "Click" event handler attached to the "Probability" filter control in Management Report Filters**

This event handler, much of whose body has been intentionally omitted, will cause an overlay selection window to appear when the user clicks on the "Probability" filter control. Functionally speaking, this event handler achieves its purpose. But, design wise, it leaves a lot to be desired.

As one can see from the selector being passed to the `on()` function, this event handler is attached to all elements whose `data-role` attribute is one of the following: `report-selection-overlay`, `mgmt-report-selection-overlay` or `prjtrack-report-selection-overlay`. This shows that the behaviour of the "Probability" filter control is shared by many other elements. By attaching the same event handler to all of these elements, a good deal of code repetition is eliminated. Nevertheless, because of subtle differences between these three classes of elements, the function must make additional checks during runtime in order to alter its behaviour accordingly.

When used sparingly, this design is acceptable as it is an expedient and relatively straight-forward solution. But a review of the `init_reports()` function shows that these types of checks are occurring over and over again. This design, in addition to being repetitive and hard to maintain, pose a major challenge to future growth and expansion. If more complex requirements and edge cases appear in the future, one can easily find themselves trapped in "If-Else Hell" with this design.

Speaking of code repetition, the author would like now to highlight the most glaring issue of the Management Report Filters' implementation. To do so, an analysis of exactly how the filters are applied is in order.

### 2.4 – A Senseless Repetition

As mentioned earlier, the init_reports() function creates a *closure* in which a variable, mgmt_report_filters, is defined (Figure 2.5).

```
var init_reports = (function() {
  // Note: List_Map_Data is a wrapper class for the regular JavaScript
  // Object. Instances of it can be treated like a regular Object.
  var report_filters = new List_Map_Data();
  var mgmt_report_filters = new List_Map_Data();
  var prjtrack_report_filters = new List_Map_Data();

  /* ... */

  return function(body) {
    /* ... */

    body.on('click',
      '[data-id="report-filter-apply"]'
      + ', [data-id="mgmt-report-filter-apply"]',
      function() {
          /* ... */

            setMgmtReportFilters(mgmt_report_filters);

          /* ... */
      });

    /* ... */
  }
})();
```

**Figure 2.5: The “init_reports()” function**

This variable holds an object that will keep track of the Management Report Filters as the user interacts with its various controls. Figure 2.6 shows an example of what mgmt_report_filters might look like after certain filters have been selected.

When the user clicks on the "Apply" button, the "Click" event handler shown in the latter half of Figure 2.5 runs. This event handler will perform all of the steps necessary in applying the filters. The author has chosen the show the only part of this function relevant to the discussions at hand: the call to the function setMgmtReportFilters().

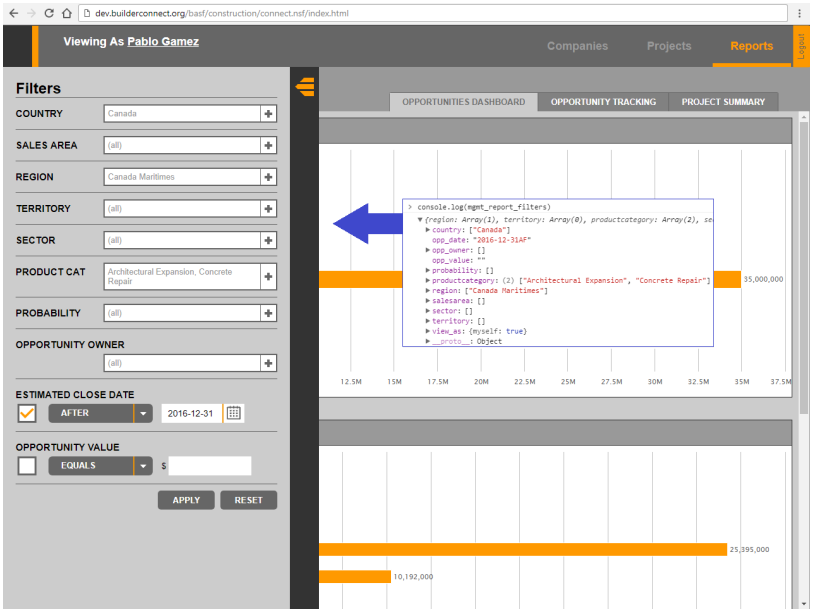


**Figure 2.6: "mgmt_report_filters" changes to reflect the filters panel**

This function has a global scope, thus allowing it to be invoked from anywhere throughout the program. However, its definition is found within the *closure* of another function, initReport() (Figure 2.7).

```
var initReport = ( function() {
  var report_filters = new List_Map_Data();
  var mgmt_report_filters = new List_Map_Data();
  var prjtrack_report_filters = new List_Map_Data();

  /* ... */

  setMgmtReportFilters = function(filters) {
    mgmt_report_filters.add('region', (filters.region || []).slice(0));
    mgmt_report_filters.add('territory', (filters.territory || []).slice(0));
    mgmt_report_filters.add('sector', (filters.sector || []).slice(0));
    mgmt_report_filters.add('productcategory', (filters.productcategory || []).slice(0));
    mgmt_report_filters.add('probability', (filters.probability || []).slice(0));
    mgmt_report_filters.add('opp_value', filters.opp_value);
    mgmt_report_filters.add('opp_date', filters.opp_date);
    mgmt_report_filters.add('opp_owner', (filters.opp_owner || []).slice(0));
    mgmt_report_filters.add('country', (filters.country || []).slice(0));
    mgmt_report_filters.add('salesarea', (filters.salesarea || []).slice(0));
  }

  /* ... */

  return function(options) {
    /* ... */
  }
})();
```

**FIGURE 2.7: The “initReport()” function and definition for “setMgmtReportFilters()”**

If the trio of variables at the beginning of the initReport() function seems familiar, it is because they are an exact replica of another trio of variables seen earlier in the init_reports() function. Furthermore, it is also clear from Figure 2.7 that the only purpose of the setMgmtReportFilters() function is to transfer data from its filters argument over to the mgmt_report_filters variables defined in the *closure* of init_reports().

Thus, in Figure 2.5, when the "Click" event handler on the "Apply" button makes a call to setMgmtReportFilters(), all that happens is that the contents of the mgmt_report_filters variable in the init_reports() function is copied over to the mgmt_report_filters variable in the initReport() function.

Further examination reveals that the setMgmtReportFilters() function is only ever called from within the init_reports() function. Additionally, in all instances, the mgmt_report_filters variable (The one in the *closure* of init_reports()) is passed as the sole argument. This demonstrates that the only purpose of the setMgmtReportFilters() function is to "extend" the scope of the mgmt_report_filters variable to the initReport() function.

This is hardly a good design. With this repetition comes the need to maintain two distinct sets of objects. As one makes changes to the Management Report Filters in the future, they must ensure that the changes are reflected in both init_reports() and initReport(). Furthermore, if anything ever breaks, this repetition also means that there is one more point which the developer might have to check in their efforts to debug.

## 3.0 – Conclusions

The Management Report Filters are a crucial component of the Management Reports. While it is functionally stable, several weaknesses in its implementation and design create unnecessary complications and hinder future development. When one seeks expediency in solving a problem or implementing a feature, the failure to follow good design principles is often the cost. For issues with limited scope, this trade-off is occasionally acceptable. But as the system grows and expands, there is always a good case to be made for going back and reviewing previous design choices to analyze their scalability for the future. Failure to do so frequently leads to convoluted and unwieldy programs that are as hard to maintain as they are to be expanded upon.