Paper SD12

Next-Generation Deliverable Tracker

Michael Byrne, Veramed, UK

ABSTRACT

Deliverable Trackers are a critical tool in biometrics workflow and are used for everything from project planning, resource management, through to QC and validation tracking, and as a quality audit record.

The standard approach is to use Microsoft Excel– this offers a well understood user interface; however it relies on manual entry that is often out of date, and data is duplicated in multiple places with limited ability to link and synchronize data from other systems.

This presentation demonstrates a new approach to implementing a deliverable tracker that uses an interactive user-interface (R-Shiny) backed with a database populated by harvesting metrics directly from the SCE and from other systems via API calls. This approach provides delivery teams with recognizable deliverable tracker user experience however significantly reduces the need for manual entry, provides live automated results and also allows project metrics to be pooled centrally to enable data-driven process improvement insights.

INTRODUCTION

It is essential in clinical trials to efficiently track and manage programming deliverables to ensure timely results without compromising on quality. Deliverable trackers provide an overview of project status, a detailed list of deliverables, and often a method for assigning responsibility for programming tasks. While traditionally teams have relied on spreadsheets for tracking projects, this paper explores how improvements can be made using an alternative approach.

New workflow tools, project management tools and version control systems are increasingly being used in clinical trials. While providing efficiencies they can also present some new challenges by introducing additional sources to maintain and keep synchronized. The paper will also explore an approach to address this potential problem.

The strengths and limitations of the traditional spreadsheet approach will be examined. This will help guide the design of a new approach—identifying which elements to retain and which can be optimized. Lastly, the paper will present an example solution featuring an R Shiny dashboard connected to a metrics database and integrated with GitLab through its API.

EXISTING TRACKERS

Spreadsheets are commonly used as the go-to method when it comes to tracking due to the ease at which they can be set up, the intuitive interface they provide and the widely understood controls for updating and interacting with the information provided. This new approach will attempt to retain these advantages, while addressing some of the main limitations that spreadsheets can present:

EFFORT

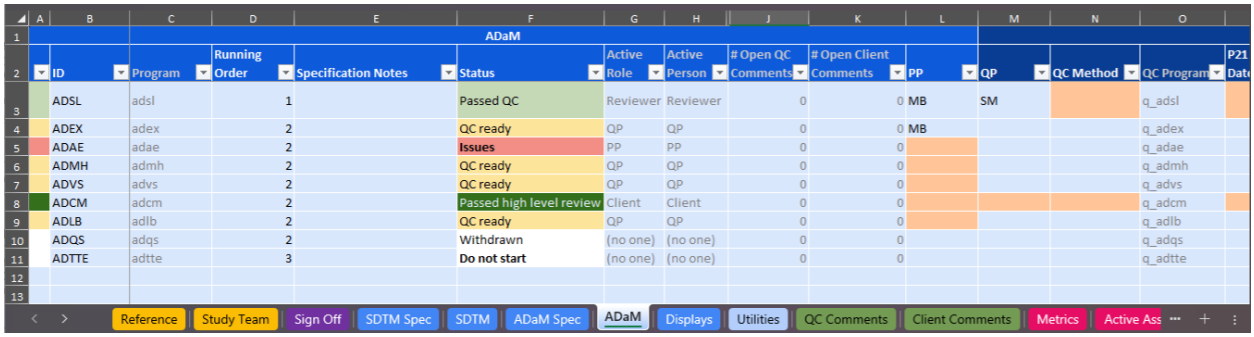
Although some level of manual entry is acceptable and even desired, there are certain information points that can be automatically populated in a future tracking solution. Some spreadsheet trackers rely on user input of data points such as the date and time that QC and production arms of a programming effort were last run. These can then be cross referenced to make sure that the correct order in the process is being followed. This information can be made available by setting up a process to capture and store it in a system accessible to the tracker. Duplication of effort can also be a factor when using multiple systems alongside a spreadsheet tracker as some information will need to be input in more than one system. Integrating a tracker with these other systems negates this additional effort.

QUALITY

Manual population of fields within the tracker is also a point where errors can occur. Automatically populating certain information within the tracker not only cuts down on quality issues of those data points but also allows automated checks to be set up to give even more confidence in the process. In the example discussed above, automatically populating program run timestamps would enable automated checks to be set up that can flag things such as programs not being run in the correct order or programs not being run since an update has been requested. Integrating the tracker with other systems will improve quality as it will enable there to be a single source of truth for data points that were previously being recorded manually in multiple places.

CURRENCY

When dealing with a manually populated tracker it can be difficult to know if the information being viewed is up to date. This problem is exacerbated with larger team sizes as it becomes increasingly difficult to know whether everyone has updated the information relative to their portion of the work being tracked. Linking a tracker to other systems directly ensures that information being viewed is up to date.



	ID	Program	Running Order	Specification Notes	Status	Active Role	Active Person	# Open QC Comments	# Open Client Comments	PP	QP	QC Method	QC Program
	ADaM												
3	ADSL	adsl	1		Passed QC	Reviewer	Reviewer	0	0	MB	SM		q_adsl
4	ADEX	adex	2		QC ready	QP	QP	0	0	MB			q_adex
5	ADAE	adae	2		Issues	PP	PP	0	0				q_adae
6	ADMH	admh	2		QC ready	QP	QP	0	0				q_admh
7	ADVS	advs	2		QC ready	QP	QP	0	0				q_advs
8	ADCM	adcm	2		Passed high level review	Client	Client	0	0				q_adcm
9	ADLB	adlb	2		QC ready	QP	QP	0	0				q_adlb
10	ADQS	adqs	2		Withdrawn	(no one)	(no one)	0	0				q_adqs
11	ADTTE	adtte	3		Do not start	(no one)	(no one)	0	0				q_adtte

Reference | Study Team | Sign Off | SDTM Spec | SDTM | ADaM Spec | ADaM | Displays | Utilities | QC Comments | Client Comments | Metrics | Active Ass

Figure 1 - Example of a Spreadsheet Deliverable Tracker

SOLUTION DESIGN

With the increasing use of workflow tools, version control systems and project management tools in clinical trials it's becoming increasingly important to be able to integrate different tools and systems with one another. This ability to integrate allows us to select the best tool for each task instead of needing to build one tool that does everything. The solution that's been developed has been designed to integrate the main tracker dashboard with GitLab to take advantage of its version control and issue tracking capabilities along with a database for programming metrics and for tracking QC status.

Figure 2 - Solution Process Layout

SCE METRICS

One feature of Statistical Computing Environment which is becoming increasingly important is the ability to capture and report essential business-related metrics. This helps to track and to identify possible opportunity areas for optimization. While some of this information is starting to be made available in modern SCEs it's still necessary to explicitly collect certain data points that are of specific interest. In this solution the metrics of particular value were

those detailing what programs have been run and when they were last run. The collection of this information was achieved by the addition of some standard code which is included by every program. The additional code in this case writes out a metadata file to a central location each time a program is run. The file contains key information about the program execution including the user that ran the program, the time the program execution started and stopped and the program name and location. These metadata files are then collected on a set frequency by an automated process and loaded into a database. In addition to this there is a set of scripts that are run on a set frequency to provide more detailed information about program contents, program logs, macros used and disk usage.

Figure 3 - SCE Metrics Harvesting

Collecting this information as programs are executed means that the database can be integrated with the tracker dashboard and used as a reliable and up to date source of several key pieces of information. We can now automatically populate the tracker with current information on which programs exist, when each program has last been run; whether an execution had any errors, warnings or notes of interest and even which macros are being used by each program.

GITLAB INTEGRATION

GitLab was chosen as the tool to integrate with in the solution design as it was already being used for version controlled projects at the company. GitLab has an API which allows users to interact with GitLab programmatically. As we're using R Shiny for the tracker user interface, an R package "gitlabr" was chosen as it provides functions for many of the common requests used in interacting with GitLab. Integrating the dashboard with GitLab allowed many useful functions to be performed from the dashboard itself. A few examples used in our solution:

- Development branches are set up for each program in the tracker with the click of a button.
- The list of users assigned to a project in GitLab is retrieved and used to populate drop-down lists of available users in the modal dialogs within the dashboard.
- Issues are created from the dashboard and assigned to available users and linked to a particular program using a standard prefix.
- Merge Requests are set up from the dashboard with options for setting assignees and reviewers. The issues that the merge requests close can also be selected from a list of all issues linked with a particular program so that the correct closing text will automatically populate in the description field.

Figure 4 - Example gitlabr functions

R SHINY DASHBOARD

Shiny is an R package that makes it easy to build interactive web apps directly from R code. The use of Shiny for the deliverable tracker means that it can be developed and maintained more easily by statistical programmers as many of them are already familiar with the R language. Dashboards built using Shiny are reactive; they can be programmed to automatically update the displayed data and visuals based on user inputs such as sliders, drop down menus and buttons.

This reactivity makes Shiny an excellent choice for building a tracker user interface where users can make changes easily and view the real-time updates. The use of Shiny to build the tracker also allows a great deal of control over the appearance of the dashboard. This made it possible to keep a lot of the familiarity of the spreadsheet layout in the design which is helpful to users adapting to a new system. The example below shows how the tabular layout of a spreadsheet-based tracker has been closely replicated by outputting a table and adding buttons to columns for performing actions related to each program in the table.

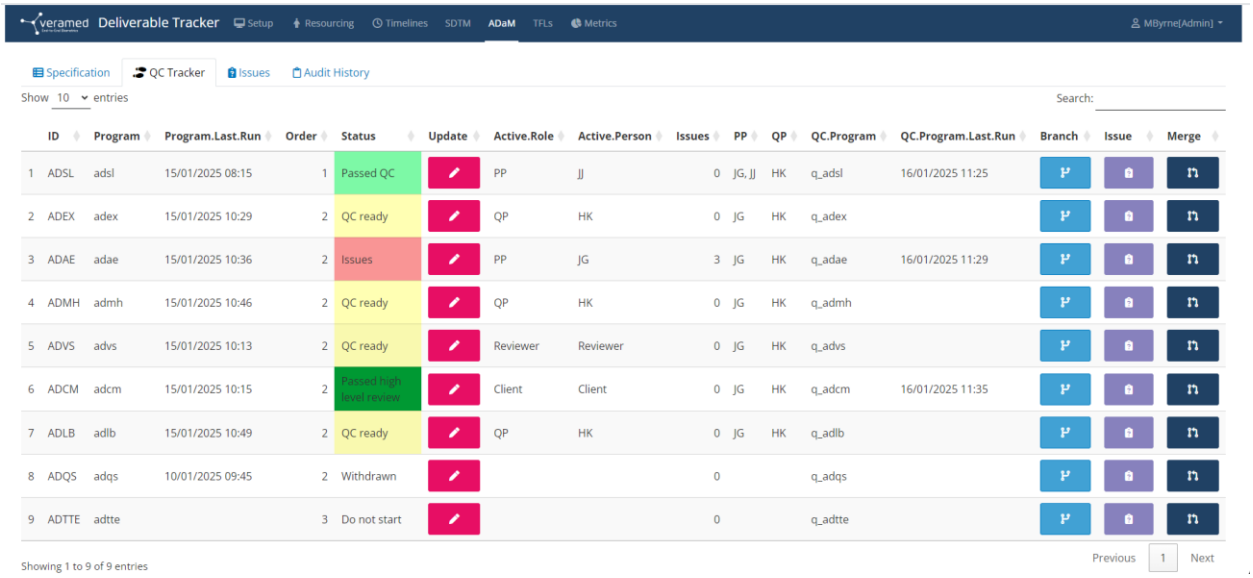


Figure 5 - R Shiny Dashboard for Deliverable Tracker

The buttons added to the data table can be programmed to perform tasks related to the programs. For this solution the branching strategy was kept quite simple - a user can create a development branch for a particular program and then a merge request for this branch can be created once the QC is at a point where the program has passed all review stages. Issues can also be set up from the dashboard and are linked to specific programs by using a standard prefix of the program ID. This way issues can be created and grouped together based on certain rules used when retrieving issues and branches from the GitLab API.

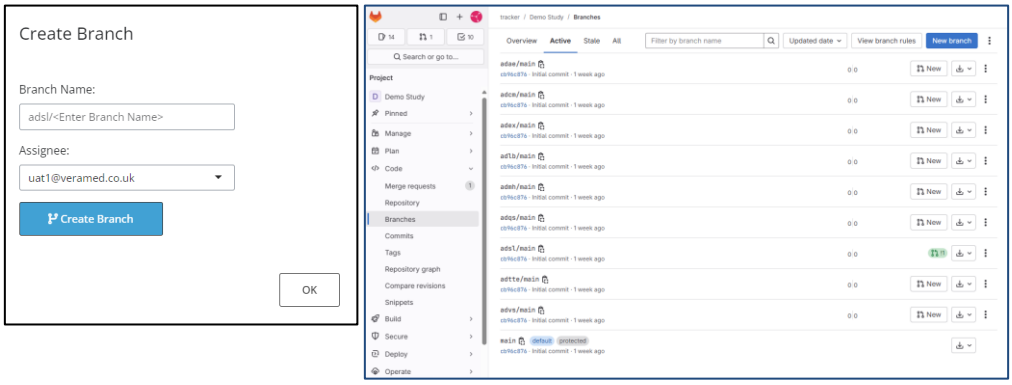


Figure 6 - Example of a programmed button updating information in GitLab project

AUDIT HISTORY

Fields that would have been updated manually in a spreadsheet tracker can now be updated using a programmed button and dialog form. As an example, the status field above has a button in the adjacent column which can be clicked to bring up a dialog form that allows a user to select from a pre-defined list of status values and provide an accompanying comment. These values are written to a database table along with information about who performed the action and when it was performed. This enables an audit history of all user operations to be stored in a database which can be used to provide reporting or combined with other information metrics.

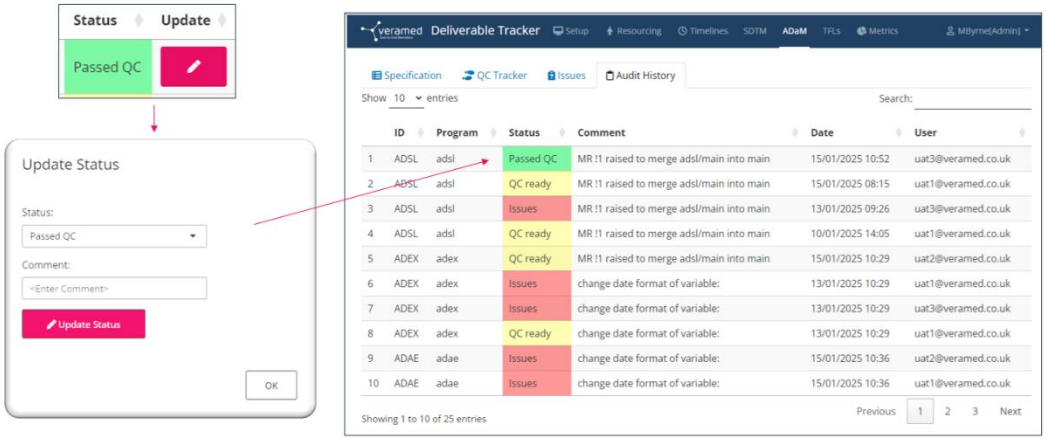


Figure 7 - Example of a programmed button displaying a form to input data

CONCLUSION

This paper demonstrates that we can reduce the amount of manual effort, improve quality and ensure information is more up to date by moving away from the traditional spreadsheet tracker. The solution developed uses an R Shiny dashboard that integrates with an SCE metrics database and interfaces with GitLab's API. This approach provides a single source of truth without requiring all information to be stored in the same system. The design offers a familiar user experience which will ease the transition to a new system. The audit history that this approach provides will offer valuable insights at the company level, helping to inform future decisions related to estimates and process improvements.

REFERENCES

[1] GitLab REST API documentation - https://docs.gitlab.com/ee/api/rest/

[2] gitlabr R package documentation - https://cran.r-project.org/web/packages/gitlabr/index.html

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Michael Byrne
Manager, Standards Programming
Technology Solutions Group
Veramed
Michael.Byrne@veramed.co.uk
http://www.veramedinc.com