Documentation Automation for the Verification and Validation of Rubin Observatory Software

Gabriele Comoretto[1], Leanne P. Guy[1], William O'Mullane[1], Keith Bechtol[1,2], Jeffrey L. Carlin[1], Jonathan Sick[1], Brian Van Klaveren[3], and Austin Roberts[1]

[1]Rubin Observatory Project Office, 950 N. Cherry Ave., Tucson, AZ 85719, USA
[2]Department of Physics, University of Wisconsin-Madison, Madison, WI 53706, USA
[3]SLAC National Accelerator Laboratory, 2575 Sand Hill Rd., Menlo Park, CA 94025, USA

January 20, 2021

ABSTRACT

The Data Management (DM) subsystem of the Vera C. Rubin Observatory Legacy Survey of Space and Time (LSST) is responsible for creating the software, services, and systems that will be used to produce science-ready data products. The software, currently under development, is heterogeneous, comprising both C++ and Python components, and is designed to facilitate both the processing of the observatory images and to enable value-added contributions from the broader scientific community. Verification and validation of these software products, services, and systems is an essential yet time-consuming task. In this paper, we present the tooling and procedures developed to ensure a systematic approach to the production of documentation for verification and validation. By adopting a systematic approach, we guarantee full traceability to system requirements, integration with the project's Systems Engineering model, and substantially reduce the time required for the whole process.

1. INTRODUCTION

In its first 10 years of observations, the Rubin Observatory will perform the Legacy Survey of Space and Time (LSST), as described in Ref. 1. In the construction phase of the observatory, the Data Management subsystem, described in Ref. 2, is in charge of developing and preparing the software, services, and systems that will automatically reduce the raw data to produce science-ready data products. Project requirements on DM products are documented in change-controlled specifications. DM Verification and Validation(V&V) activities are planned to guarantee that survey software and infrastructure both fulfill the system requirements and enable the science that motivates the project.

We present here the tooling and procedures used at the Rubin Observatory for the documentation of verification and validation activities. These are an evolution of the approach to verification and validation adopted by the Gaia Data Processing and Analysis Consortium (DPAC), described in Ref. 3. All test activities are managed using the Jira issue tracking and project management tool. Test cases are created and updated, and test results are reported using Jira. This ensures that all elements to be documented (e.g. requirements, verification elements, etc) are available through one tool, and that their history is maintained. The project's MagicDraw based systems engineering model is described in Ref. 4. The requirements definitions in MagicDraw are synced with the Jira test framework, providing a direct link between tests and system requirements. Test documents can therefore be generated automatically, avoiding typical problems such as a lack of traceability, typographical errors, duplication of content, and misalignment between documents. The centralized collection of information permits a high level of automation, where the extraction of test documents is achieved by a continuous integration process. This systematic approach substantially reduces the time required to produce verification and validation documentation, improves integration with the project's system engineering model, and ensures full traceability to system requirements. Such an approach is all the more necessary when considering the number of requirements that need to be verified and documented; approximately 700 for Data Management alone.

To complete the picture, the following provides a description of each document involved in the process:

- *Test Specification* documents to baseline the test cases,

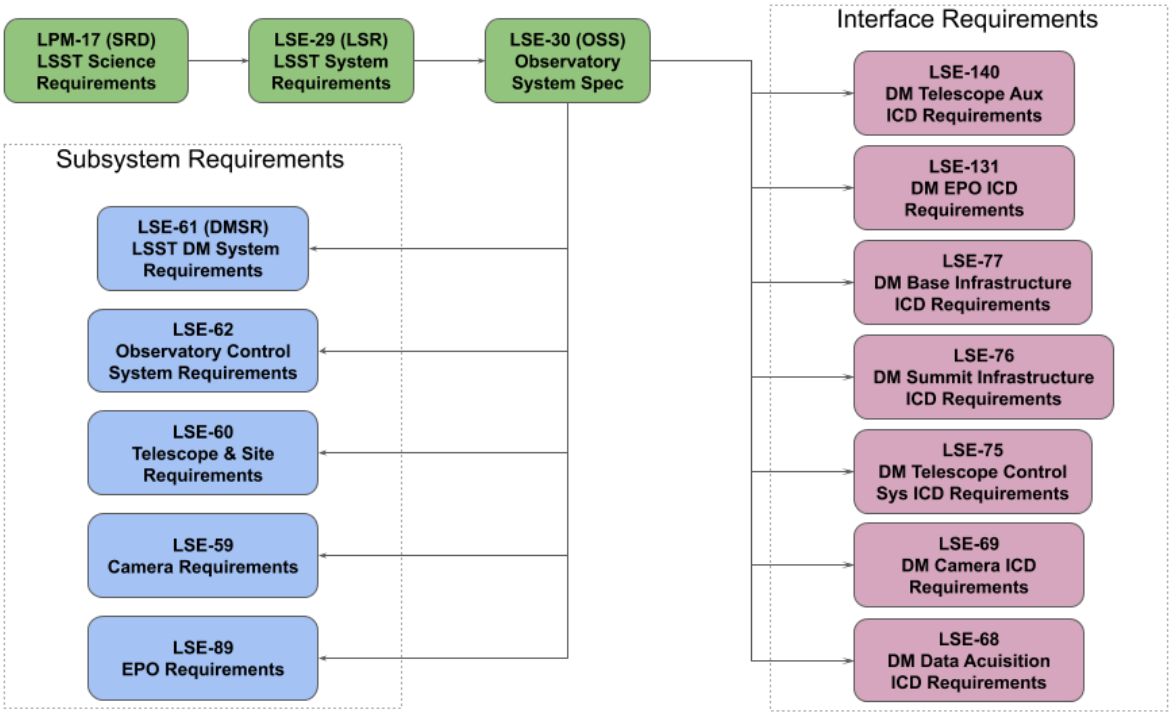

Figure 1. The Rubin Observatory top level document tree. The top-level requirements documents are in green, the LSST subsystem requirements documents in blue, and the DM related interface requirements documents in red.

- *Verification Element Baseline* documents to baseline the verification elements,
- *Test Plan and Report* documents that provide test campaign planning and reporting information,
- A *Verification Control Document* that, using the same approach and information already available in Jira, becomes a useful document to summarize the internal progression and completion to the requirements coverage, and an important document to present to funding agencies and at project reviews.

1.1 Organization of this Paper

Section 2 introduces the Rubin Observatory approach to requirements verification. Section 3 describes the adopted approach including a detailed discussion of the tools and procedures developed, and the documents produced. The Verification Control Document, which provides a summary of the status of all verification and validation activities is presented in Section 4. In closing, a few considerations on the implemented approach and outcomes are discussed in Section 5

2. REQUIREMENTS VERIFICATION

The goal of verification and validation is to ensure that all requirements have been implemented according to specification, and where not, identify deviations and non-conformances. Figure 1 shows the flow down of requirements from high level requirements documents to each of the subsystems. The Data Management subsystem requirements, which are flowed down from the LSST Observatory System Specifications (OSS),[5] are baselined in the Data Management System Requirements document (DMSR).[6] Interface requirements between DM and other Rubin Observatory subsystems also impact Data Management delivered components and, therefore, need to be considered during Verification and Validation.

DM is responsible for both fully verifying the DMSR requirements and contributing to the verification and validation of those interface requirements that have an impact on the DM subsystem. To achieve this, we create

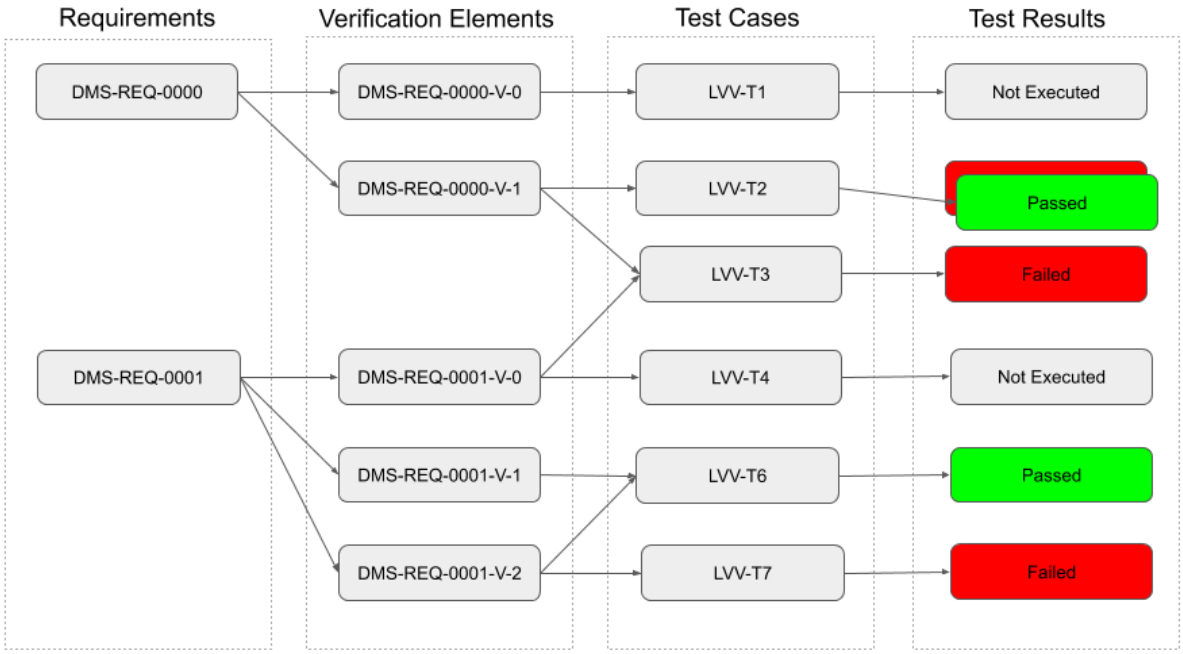

Figure 2. Schematic approach to the Verification and Validation.

one or more verification elements for each and every requirement. These verification elements are assigned to the DM validation scientist who will ensure they are properly described and are sufficient to fully verify the corresponding requirements. In the case that the initial set of verification elements are not sufficient to verify a requirement, the validation scientist will request additional verification elements to be associated with the requirement. Conversely, if an initial verification element is not needed, it will be removed. Figure 2 shows the relationship between requirements, verification elements, test cases and results.

All verification and validation activities are organized into test campaigns, each of which is associated with a single milestone*, The results of the test executions are reported in Jira and coverage information is propagated back to the verification elements and requirements in the system engineering model. Test documents, including test reports, are generated automatically from Jira. This approach ensures that all elements required for the tests are linked and available in Jira, providing traceability and making it possible to extract the necessary information into test documents without manual intervention.

3. THE VERIFICATION AND VALIDATION APPROACH

The verification and validation processes, methodologies and tools as illustrated in Ref. 4 have been implemented. Required tools like *Syndeia* and *Docsteady* have been put in place. Others tools used in the process described in this section, are generally used in other project activities.

3.1 Tooling

Figure 3 provides a schematic overview of the tools used as part of the Verification and Validation process. Figure 4 shows a few screenshots of the tools as implemented at Rubin Observatory. We can see how Verification and

*A milestone is a set of functionalities or activities, that are expected to be implemented or completed at a specific point in time. In the context of V&V, milestones are defined as Jira issues. Their definition and management is not within the scope of this paper.

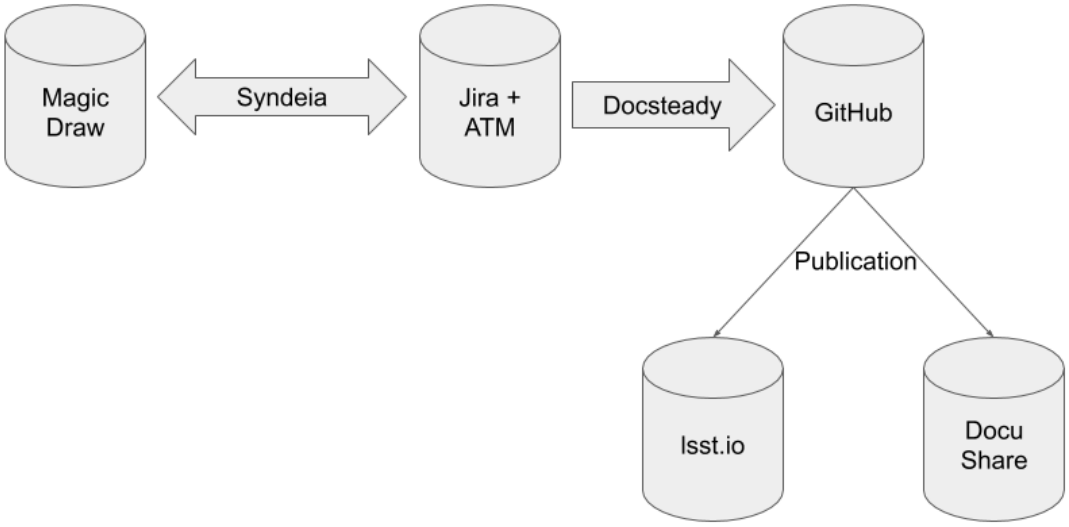

Figure 3. Schematic overview of the tools involved in the V&V process.

Validation information flows from Jira, where it is created and maintained, to the final document ready to be uploaded to DocuShare, the official Rubin Observatory documentation repository.

MagicDraw is the Rubin Observatory modeling tool. It maintains the project models and requirements, using a Model Based Systems Engineering (MBSE) approach. The verification elements are created in MagicDraw using the requirements as a starting point, and are synchronized to and from **Jira** using **Syndeia**. This ensures that the proper traceability between requirements and verification elements is in place, and that changes in one part of the system can be propagated through other components.

Jira is the Rubin Observatory issue tracking system. The *Test Manager for Jira* plugin (TM4J), previously known as *Adaptavist Test Manager* (ATM), provides additional functionality to Jira to manage test activities. Jira hosts all test information, providing an easy-to-use graphical interface to the user.

Syndeia is a third party tool that integrates **MagicDraw** and **Jira**. First the verification elements are created in MagicDraw and then the data is synced to Jira via a mapping between the fields in MagicDraw and the fields in Jira, utilizing the REST API. After the test activity is completed, the information will be synced back to MagicDraw.

Docsteady developed by the DM subsystem, is the tool that generates the test documents, extracting the information from Jira using the REST API. It can be executed manually – from the command line – or automatically – in a continuous integration (CI) tool – and generates the documents in LaTeX format. Using a CI system such as Jenkins, verification and validation documents can be generated continuously, or on a regular basis. The extracted LaTeX code is managed in Git repositories, one for each document. Docsteady's source code is available at https://github.com/lsst-dm/docsteady.

Figure 4. Tools workflow overview: bringing information from Jira *TM4J* to the published document available on the web. The information in Jira is synched with the system engineering model in MagicDraw.

GitHub is the platform used for software version management. Verification and Validation documents are written in LaTeX and maintained in Git repositories. Whenever an author pushes changes to GitHub, a cloud-based continuous integration service (typically Travis CI) renders a pdf of the document and uploads it to **LSST the Docs**.

LSST the Docs is the Rubin Observatory documentation hosting platform, see The LSST the Docs Platform for Continuous Documentation Delivery.[7] LSST the Docs hosts not only the accepted version of a document on its own subdomain of `lsst.io`, but also draft and historical versions corresponding to Git branches and tags. By publishing draft updates of documents in near-real-time, teams can collaborate and comment on documents without having to compile them locally. LSST the Docs hosts any type of static web content. To specifically publish pdf documents, we run a tool within the continuous integration service called *Lander* (available at `https://github.com/lsst-sqre/lander`) that generates an HTML landing page for the pdf based on metadata extracted from the document's source.

DocuShare is the official Rubin Observatory document repository for the construction project.

3.2 Process Overview

All Verification and Validation activities are driven by milestones, each of which describes the major DM functionality expected to be available and the associated list of requirements. We assume in this document, that the requirements have been properly formalized, documented, and approved, see Ref. 4. As described early in this paper, one or more verification elements are created for each requirement. When first created, the verification elements have no description; the only information they contain is the details of the requirements from which they were generated. They are synced to Jira using Syndeia, after which the validation scientist or a designee

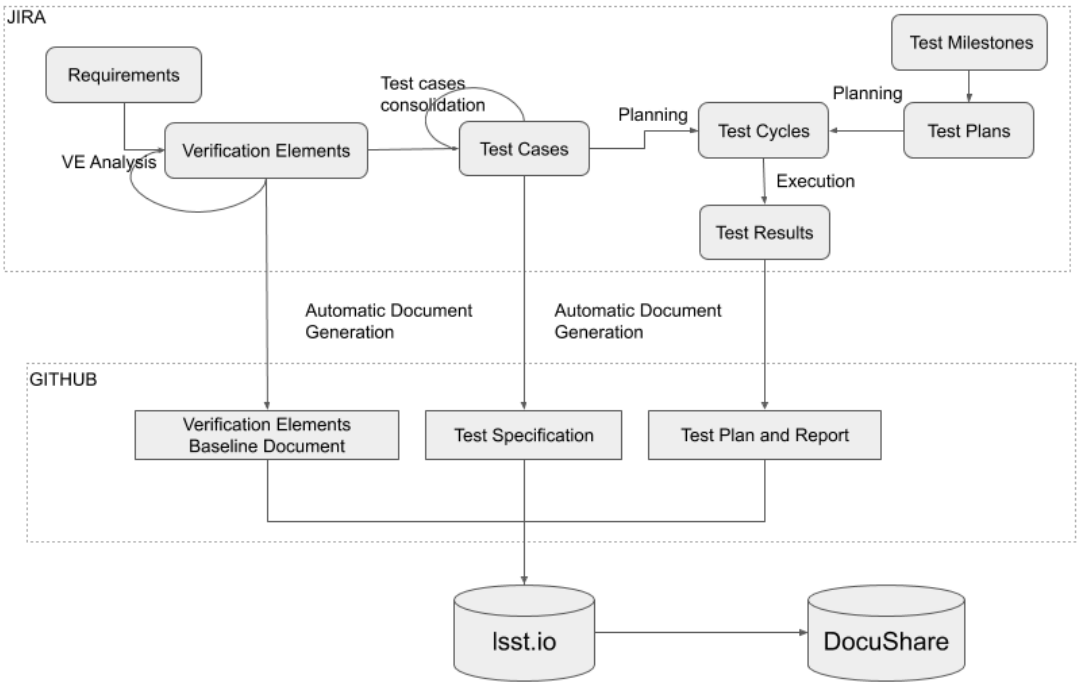

Figure 5. Schematic overview of the V&V procedure.

ensures that they are properly addressed, as already mentioned in section 2. Figure 5 gives a graphical overview of the process.

The following subsections describe the main steps of the verification and validation activities that take place in Jira.

3.2.1 Verification Elements Analysis

The first activity required before starting with any test campaign, is to ensure that each verification element is fully described.

In this phase, the verification elements shall be completed with the following information:

- the description, which clarifies the scope of the verification element, that is, which part or parts of the requirement will be verified.
- one or more draft test cases, each of which are defined with a one-sentence objective and an owner. In this way each test case is automatically traced to a verification element and therefore to a requirement.

Verification elements are baselined in a *Verification Baseline Document*.

3.2.2 Test Case Consolidation

The owner of each test case is responsible for completing it and providing all relevant information in Jira , such as, for example the test objective, the pre- and post-conditions, and the test procedure including the steps to execute during the test campaign. Additional test cases can be created as needed. All test cases are related to a verification element in order to guarantee the traceability. In general, test cases can be associated with multiple verification elements, and therefore with more than one requirement.

Test cases should be general, specifying only the software component or the dataset type required. The exact software and/or dataset versions and the configuration to be used during the test is provided in the test campaign planning phase. Test cases are baselined in *Test Specifications*.

3.2.3 Planning and Execution

As described in Section 2 all test activities are organized into test campaigns. For each test campaign, two *TM4J* objects are created:

- **TM4J Test Plan**: provides the context of the test activity, and typically corresponds to a project milestone.
- **TM4J Test Cycle**(s): provides the subset of tests to be executed. For each test campaign we may have multiple test cycles, with different configurations, datasets, or extra conditions we may want to test. Each Test Cycle is traced to the TM4J Test Plan, and provides the list of test cases that need to be executed within that scope.

For each test campaign we identify two phases:

Planning This is the preparation phase of the test campaign. All relevant information, e.g. software version, datasets to use, hardware or configuration, is collected in the two *TM4J* objects defining the test campaign. At the successful completion of this phase the test is ready to start.

Rubin Observatory does not instigate a formal Test Readiness Review for each test campaign, however the tooling and procedures in place permit the person who is responsible for the test activity, and relevant stakeholders, to assess and review the readiness of the activity before the execution of the tests. This is done by extracting the information into a document, the *Test Plan and Report*, in GitHub, generating the pdf and making it available in the corresponding LSST the Docs landing page. The Product Owner, contributors and stakeholders can easily access the document, check the content, and comment, ask for clarification, or request changes using the GitHub Pull Request (PR) mechanism or the corresponding Jira issue. When agreement has been reached, the TM4J Test Plan status is changed to **Approved** to indicate the test activity is ready to start.

The approved *Test Plan and Report* document resulting at the end of this first phase, is uploaded to DocuShare and used as a reference for the next phase. The document at this stage provides the agreed test procedures and all the necessary information required to execute the tests.

Execution In this phase, the testers, identified in the previous phase, execute the test procedures and document the result of each step in Jira. The *TM4J* plugin provides a test player view, where for each step in each test case, it is possible to say if it has been executed successfully or not, specifying the result of the step, and associating any Jira issues that have been found during the test execution.

The test result information is extracted and added to the *Test Plan and Report* document. Once the test execution is complete, an overall assessment must be provided in the *TM4J* Test Plan.

As in the previous phase, the document is generated automatically from Jira and provides an easy way to access the test campaign information. Product owner, contributors and stakeholders can review the outcome of the test campaign using the same GitHub PR mechanism reported above, commenting, asking for more information, or changes if required. Finally, the Product Owner has the final sign-off on the results of the test campaign.

The *TM4J* Test Plan status will then be changed to **Done**. At the end of the test campaign, the final *Test Plan and Report* is issued and uploaded to DocuShare. We can assume that, in special cases, the document can be used to document further executions in the context of the same test campaign. For example, if the test campaign verifies a specific software release, a future patch on that software release can be tested with the addition of a specific Test Cycle to the *TM4J* Test Plan. However, future test campaigns related to different milestones will be documented in a different document.

3.3 Test Documents

The documents identified in the above steps are generated using Docsteady. The generation can be done manually or automatically. Automation is particularly useful if we want to see daily progress published on the LSST the Docs landing page.

The extraction tool ensures homogeneity of information at all levels in the Verification and Validation activities, and allows the user to concentrate only on the relevant test activities, without the need to worry about any documentation aspects.

The test documents are:

- **Test Specification**: baselines the test cases. A new issue of the document is made each time there is a substantial change in the definition of test cases.
- **Test Plan and Report**: for a single test campaign this includes all planning and execution information. It is issued at least twice. The first issue corresponds to the consolidation of the planning activity, the second to the finalized test campaign containing the results of the same.
- **Verification Baseline Document**: provides a snapshot of the verification elements. The verification elements are maintained as Jira issues, and thus are easy to change. Having a snapshot of the verification elements at a specific point in time (in the form of a document) makes for a stable baseline and is good for tracking changes. Approved versions are uploaded in DocuShare and used as a reference in other test documents.

Test Specifications and Verification Baseline documents are created for specific components, where components are the main products in a subsystem.

4. THE VERIFICATION CONTROL DOCUMENT

The Verification Control Document (VCD) provides an overview layer that summarize all V&V activities. Therefore it is an important management tool for assessing the level of verification and validation that has been achieved to date. It is also useful to include it in review documents packs, to show which requirements have been verified.

Given that all information pertaining to V&V activities is stored in Jira, the VCD can easily be extracted on-demand using the document generation tool Docsteady. As for all other test documents, the VCD is a LaTeX document, managed in a GitHub repository, built with Travis CI and published through LSST the Docs.

Given that both the number of requirements and verification elements for Rubin Observatory is very high – approximately 700 requirements and 1000 verification elements in DM alone – a separate VCD is generated for each subsystem.

There are two main sections to the VCD:

- **Summary Information** where a status overview is given. This includes the number of requirements and verification elements covered by passed (successfully executed) test cases. Figure 6 shows an example extracted from the DM VCD.
- **Detailed Information** where for each requirement, the verification elements, test cases and the status of the test cases are shown. Links to the test documents are provided, but not descriptive information. Figure 7 shows a detail example extracted from the DM VCD.

	Priority	Fully Verified 2.1.1	Partially Verified 2.1.2	With Failures 2.1.3	Not Verified 2.1.4	Not Covered 2.1.5	Total
DM Requirements	(All)	17	102	4	327	260	**710**
LSE-61	1a	7	24	1	23		**55**
	1b	8	22	3	74	4	**111**
	2		4		42	6	**52**
	3		1		5		**6**
	(All)	15	51	4	144	10	**224**
LSE-68	Not Set					19	**19**

Figure 6. Example summary extracted from the DM VCD. Here we see summary for the DMSR[6] and for the LSE-68 interface requirements specification.[8]

DMS-REQ-0043 LSE-61 (p. 1a)	DMS-REQ-0043-V-01 LVV-18 (p. Not Specified)	LVV-T21 LDM-533	2019-05-22 DMTR-53 LVV-P44	**Passed**
		LVV-T22 LDM-533	2019-05-22 DMTR-53 LVV-P44	**Passed**
		LVV-T129 LDM-639		**Not Executed**

Figure 7. Example of a detailed information for a requirement extracted from the DM VCD showing a verification element and three test cases, of which two have been executed and passed.

5. CONCLUSIONS AND OUTLOOK

In the context of a large project such as the Rubin Observatory construction project, with a large number of requirements, subsystems and components, the presented approach shows how integrating custom tools like Docsteady and Syndeia into the existing infrastructure significantly reduces the effort required to produce verification and validation documentation. This permits the team to concentrate on the relevant test activities, maximizing the scientific outcome.

At the same time, automation and continuous integration of the document generation makes test information much more accessible to the users and stakeholders. The tooling also ensures that aspects such as traceability between tests and requirements, homogeneity of the information or reusability of test cases are easy to implement and provide added value to the process.

Finally, the VCD provides a global overview of the verification status of all DM requirements, ensuring all DM direct and interface requirements are covered.

This approach is not only used by DM, but also by other Rubin Observatory subsystems that are facing the same challenge. The VCD for the time being is only provided by DM, but the tooling and procedures are ready to be used by all other Rubin Observatory subsystems. Using the same tools permits everybody to use the same template and have the same documentation format across the project.

ACKNOWLEDGMENTS

This material is based on work supported in part by the National Science Foundation through Cooperative Agreement 1258333 managed by the Association of Universities for Research in Astronomy (AURA), and the Department of Energy under Contract No. DE-AC02-76SF00515 with the SLAC National Accelerator Laboratory. Additional LSST funding comes from private donations, grants to universities, and in-kind support from LSSTC Institutional Members.

This research has made use of NASA's Astrophysics Data System Bibliographic Services.

APPENDIX A. REFERENCES

REFERENCES

[1] Ivezić et al., "LSST: From Science Drivers to Reference Design and Anticipated Data Products," **873**, 111 (Mar 2019).

[2] Jurić et al., "The LSST Data Management System," in [*Astronomical Data Analysis Software and Systems XXV*], Lorente, N. P. F., Shortridge, K., and Wayth, R., eds., *ASP Conf. Ser.* **512**, 279 (Dec 2017).

[3] Comoretto, G., Gallegos, J., Els, S., Gracia, G., Lock, T., Mercier, E., and O'Mullane, W., "The Information Management Tool (IMT) of Gaia DPAC and its potential as tool for large scale software development projects," in [*Modeling, Systems Engineering, and Project Management for Astronomy V*], Angeli, G. Z. and Dierickx, P., eds., **8449**, 186 – 195, International Society for Optics and Photonics, SPIE (2012).

[4] Selvy, B. M., Roberts, A., Reuter, M., Claver, C. C., Comoretto, G., Jenness, T., O'Mullane, W., Serio, A., Bovill, R., Sebag, J., Thomas, S., Bajaj, M., Zwemer, D., and Klaveren, B. V., "V&V planning and execution in an integrated model-based engineering environment using MagicDraw, Syndeia, and Jira," in [*Modeling, Systems Engineering, and Project Management for Astronomy VIII*], Angeli, G. Z. and Dierickx, P., eds., **10705**, 306 – 318, International Society for Optics and Photonics, SPIE (2018).

[5] Claver, C. F. and Team, T. L. S. E. I. P., "Observatory System Specifications (OSS), v18.3." LSE-30, Rubin Observatory, https://ls.st/LSE-30 (2018).

[6] Dubois-Felsmann, G. and Jenness, T., "LSST Data Management Subsystem Requirements, v8.4." LSE-61, Rubin Observatory, https://ls.st/LSE-61 (2020).

[7] Sick, J., "The LSST the Docs Platform for Continuous Documentation Delivery." SQR-006, Rubin Observatory, https://sqr-006.lsst.io (2016).

[8] Dubois-Felsmann, G., "Camera Data Acquisition Interface, v2.4." LSE-68, Rubin Observatory, https://ls.st/LSE-68 (2020).

APPENDIX B. ACRONYMS

Acronym	Description
API	Application Programming Interface
ATM	Adaptavist Test Management
AURA	Association of Universities for Research in Astronomy
CA	Control (or Cost) Account
CI	Continuous Integration
DE	dark energy
DM	Data Management
DMSR	DM System Requirements; LSE-61
DPAC	Data Processing and Analysis Consortium (Gaia)
HTML	HyperText Markup Language
LPM	LSST Project Management (Document Handle)
LSE	LSST Systems Engineering (Document Handle)
LSST	Legacy Survey of Space and Time (formerly Large Synoptic Survey Telescope)
LSSTC	LSST Corporation
LaTeX	(Leslie) Lamport TeX (document markup language and document preparation system)
MBSE	model-based systems engineering
NASA	National Aeronautics and Space Administration
PR	Pull Request
SLAC	SLAC National Accelerator Laboratory (formerly Stanford Linear Accelerator Center; SLAC is now no longer an acronym)
SQR	SQuARE document handle
SRD	LSST Science Requirements; LPM-17
VCD	Verification Control Document