

Documentation Automation for the Verification and Validation of Rubin Observatory Software

Gabriele Comoretto¹, Leanne P. Guy¹, William O’Mullane¹, Keith Bechtol^{1,2}, Jeffrey L. Carlin¹, Jonathan Sick¹, Brian Van Klaveren³, and Austin Roberts¹

¹Rubin Observatory Project Office, 950 N. Cherry Ave., Tucson, AZ 85719, USA

²Department of Physics, University of Wisconsin-Madison, Madison, WI 53706, USA

³SLAC National Accelerator Laboratory, 2575 Sand Hill Rd., Menlo Park, CA 94025, USA

May 1, 2020

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

The Rubin Observatory Data Management subsystem, as described in *Juric et al.*,¹ comprises 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 activities are planned to guarantee that survey software and infrastructure both fulfill the system requirements and enable the science that motivates the project.

In line with the verification approach adopted for the Gaia DPAC project, described in *Comoretto et al.*,² we present here the tooling and procedures used at Rubin Observatory for the documentation of verification and validation activities. Test activities are managed in Jira, where test cases are created and updated, and test results are reported. This ensures that all elements to be documented are available in one tool and that a history of the process is maintained. The Systems Engineering model is synced with the Jira test framework, providing a direct link between tests and requirements. Test documents can therefore be generated automatically, avoiding typical problems such as a lack of traceability, misspelling, 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, and its integration with the project’s Systems Engineering model, detailed in *Selvy et al.*,³ ensures full traceability to system requirements.

2. THE VERIFICATION AND VALIDATION PROBLEM

The main scope of the verification and validation activity is to ensure that all requirements have been properly implemented and verified. This includes identifying the requirements, the affected components and the verification procedures.

The Data Management requirements are baselined in the *Data Management System Requirements*,⁴ also known as the DMSR. The DM requirements are flowed down from the *LSST System Science Requirements Document* (SRD).⁵ Interface requirements between DM and other LSST subsystems also have an impact on

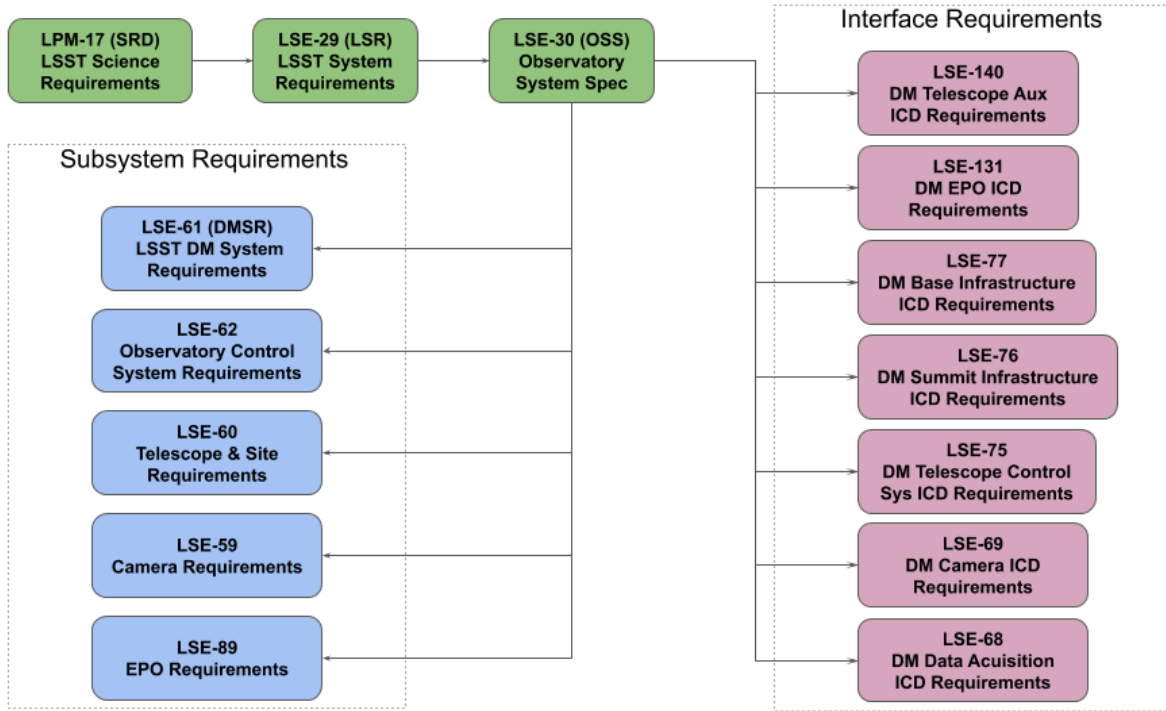


Figure 1. The Rubin Observatory top level document tree.

Data Management provided components and therefore need to be considered during Verification and Validation. Figure 1 shows the flow down of high level requirements documents.

DM is responsible for fully verifying the DMSR requirements and for contributing to the verification and validation of those interface requirements that have an impact on the DM subsystem. To achieve this, we create one or more verification elements for each requirement to be verified. These verification elements are assigned to the validation scientist, who will ensure they are properly described and are sufficient to fully verify the corresponding requirements. The validation scientist will also ensure that for each verification element there is at least one associated test case. In the case that the provided verification elements are not sufficient, the validation scientist will request additional verification elements be associated with the requirement. In the opposite case, if a provided verification element is not needed, it will be removed.

The verification and validation activities are organized into test campaigns, each of which is associated with one or more project level milestones. The results of the test executions are collected in Jira and coverage information is propagated back to the Verification Elements and Requirements. 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. Figure 2 shows the relationships between requirements, verification elements, test cases and results.

3. THE APPROACH

The verification and validation approach as illustrated in *Selvy et al*³ has been implemented. Required tools like *Syndea* and *Docsteady* have been put in place. Others tools used in the process described in 3.2, are also generally used in other project activities.

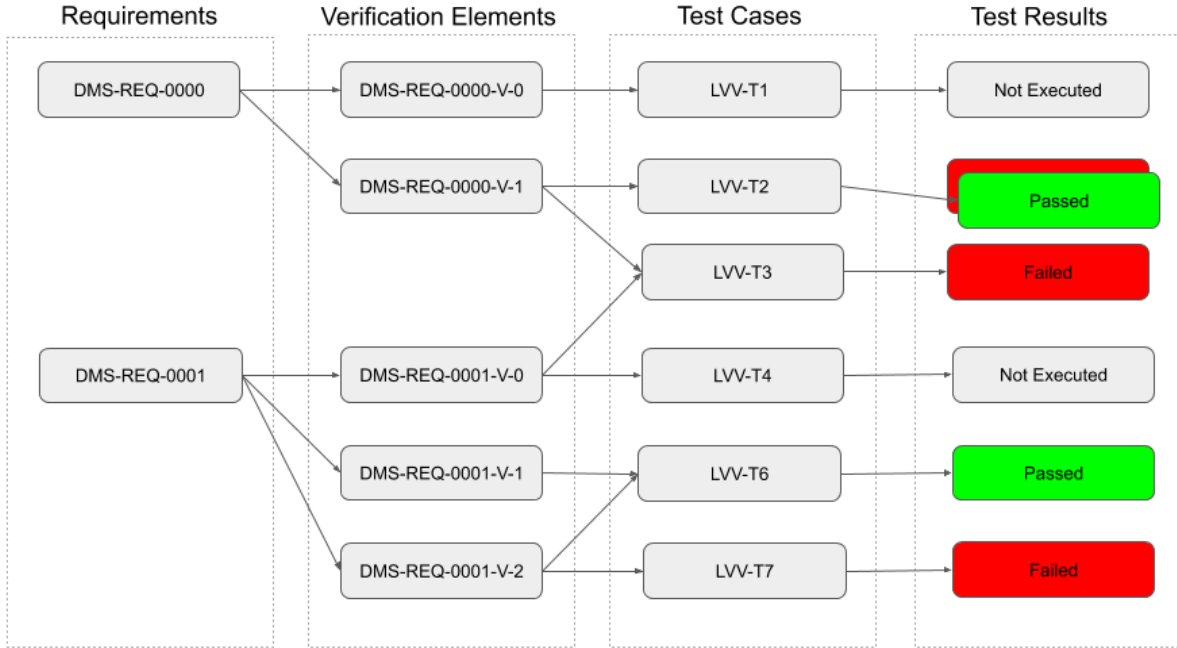


Figure 2. Schematic approach to the Verification and Validation.

3.1 The Tools Used

Here we provide a short description for the tools involved in the Verification and Validation activity. Figure 5 provides a schematic overview.

Figure 4 shows the implementation in LSST of the architecture outlined in figure 5. We can see how Verification and Validation information flows from Jira, where it is created and maintained, to the final document ready to be uploaded to Docushare.

MagicDraw is the Rubin Observatory Modeling tool. It is used for maintaining the project model, using a Model Based Systems Engineering (MBSE) approach, and for requirements management. 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.

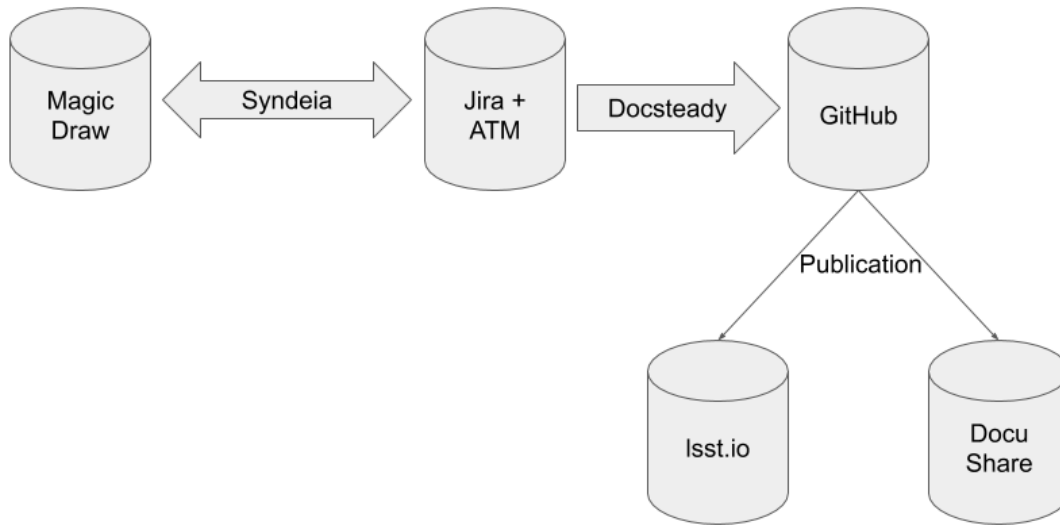


Figure 3. Schematic overview of the tools involved in the V&V process.

Docsteady 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 following the DM document workflow. Docsteady is developed within DM and the source code is available at <https://github.com/lsst-dm/docsteady>.

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 LSST’s documentation hosting platform, see *The LSST the Docs Platform for Continuous Documentation Delivery*.⁶ 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 documentation repository during construction.

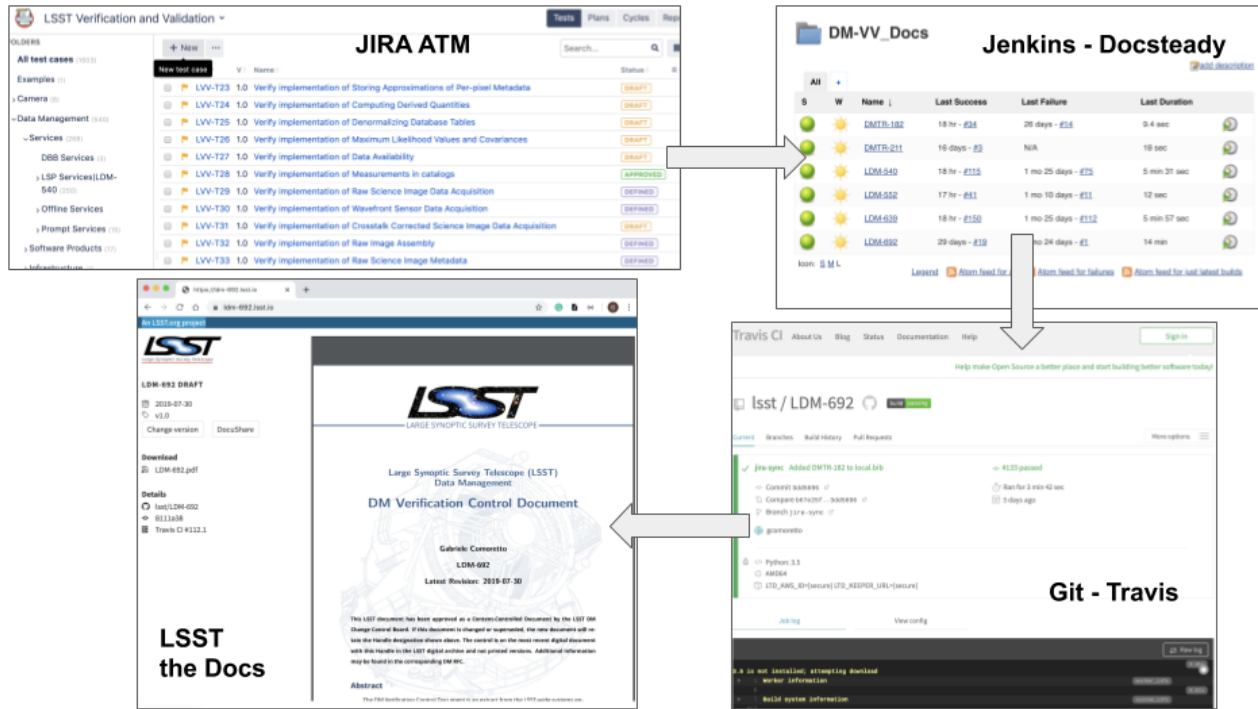


Figure 4. Tools workflow overview: bringing information from Jira TM4J to the published document.

3.2 Process Overview

All Verification and Validation activities derive from the requirements. We assume in this document, that the requirements have been properly formalized, documented and approved. As described early in this paper, a defined number of 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 requirement from which they were generated. They are synced to Jira using Syndeia, after which the validation scientist ensures that they are properly addressed. 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 complete and contains all the relevant information. This is done by the validation scientist, or delegated.

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*.

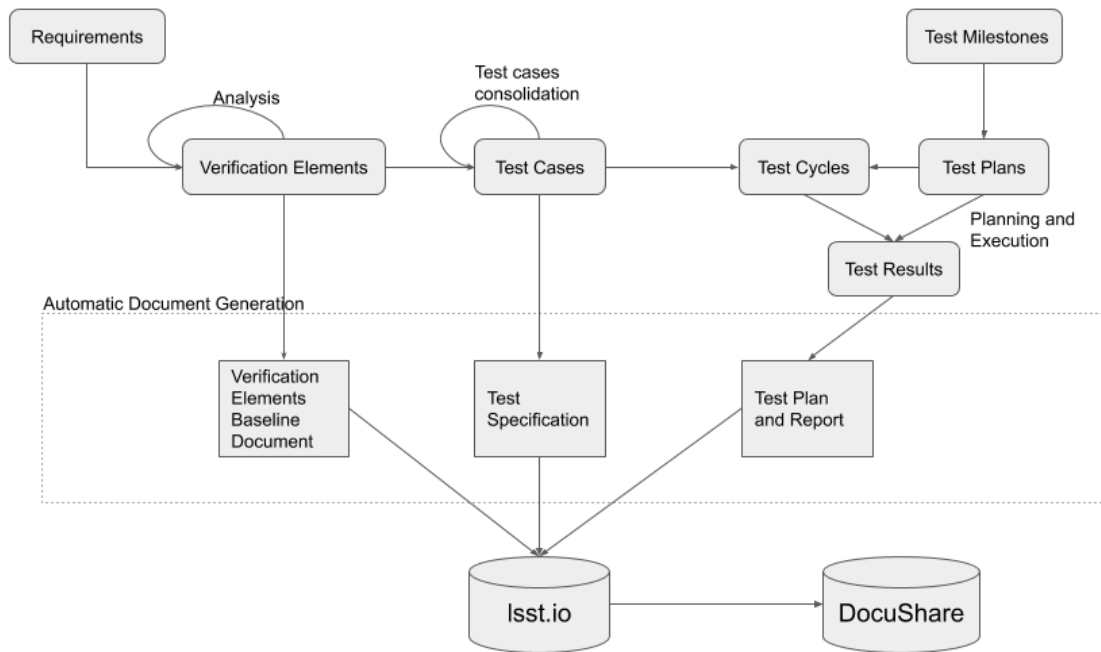


Figure 5. Schematic overview of the V&V procedure.

3.2.2 Test Case Consolidation

The owner of each test case is responsible for completing it and providing all relevant information. This includes also drafting the steps in the detailed test procedure. Additional test cases can be created, and 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 2 all test activities are organized into test campaigns. For each test campaign, two TM4J objects are created:

- **TM4J Test Plan** that provides the context of the test activity, and typically corresponds to a project milestone.
- **TM4J Test Cycle(s)** that provides the scope. 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: the Jira Test Plan and the Jira Test Cycle(s). 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 collected information. 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 produced pdf, check the content of the Test Plan, 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**. The test activity is ready to start.

The outcome of this first phase is an approved *Test Plan and Report* document to be uploaded in Docushare. 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 PR mechanism reported above, commenting, asking for more information or changes if required. Finally, Product Owner is in charge of approving the test campaign result, when s/he considers that the collected information is complete.

The TM4J Test Plan status will then be changed to **Done**. At the end of the test campaign, the Test Plan and Report is issued and uploaded to Docushare.

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 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.

	<i>Priority</i>	<i>Fully Verified</i> ^{2.1.1}	<i>Partially Verified</i> ^{2.1.2}	<i>With Failures</i> ^{2.1.3}	<i>Not Verified</i> ^{2.1.4}	<i>Not Covered</i> ^{2.1.5}	<i>Total</i>
DM Requirements	(All)	10	100	3	331	261	705
LSE-61	1a	1	25	1	26		53
	1b	7	23	2	74	4	110
	2		4		39	7	50
	3		1		5		6
	Not Set					2	2
	(All)	8	53	3	144	13	221
LSE-68	Not Set					19	19

Figure 6. Example summary extracted from the DM VCD. Here we see summary information for the DMSR⁴ and for the LSE-68 interface requirements specification.⁷

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 summarizes 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 be easily extracted on-demand using the document generation tool Docsteady. As for all other test documents, the VCD is a L^AT_EX 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 2 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 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.

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 3 test cases, of which 2 have been executed and passed.

5. CONCLUSIONS AND OUTLOOK

In the context of a large project like Rubin Observatory construction, with a large number of requirements and subsystems/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 verification control document provides a global overview of all DM requirements, ensuring we are not forgetting anything. This approach is not only used by DM, but also by other Rubin Observatory subsystems, that are facing the same challenge. Using the same tools permit everybody to use the same template and have the same documentation format across the project.

APPENDIX A. REFERENCES

REFERENCES

- [1] Jurić, M., Kantor, J., Lim, K. T., Lupton, R. H., Dubois-Felsmann, G., Jenness, T., Axelrod, T. S., Aleksić, J., Allsman, R. A., AlSayyad, Y., Alt, J., Armstrong, R., Basney, J., Becker, A. C., Becla, J., Biswas, R., Bosch, J., Boutigny, D., Kind, M. C., Ciardi, D. R., Connolly, A. J., Daniel, S. F., Daus, G. E., Economou, F., Chiang, H. F., Fausti, A., Fisher-Levine, M., Freemon, D. M., Gris, P., Hernandez, F., Hoblitt, J., Ivezić, Z., Jammes, F., Jevremović, D., Jones, R. L., Kalmbach, J. B., Kasliwal, V. P., Krughoff, K. S., Lurie, J., Lust, N. B., MacArthur, L. A., Melchior, P., Moeyens, J., Nidever, D. L., Owen, R., Parejko, J. K., Peterson, J. M., Petravick, D., Pietrowicz, S. R., Price, P. A., Reiss, D. J., Shaw, R. A., Sick, J., Slater, C. T., Strauss, M. A., Sullivan, I. S., Swinbank, J. D., Van Dyk, S., Vujčić, V., Withers, A., and Yoachim, P., “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).
- [2] 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).
- [3] 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).

- [4] Dubois-Felsmann, G. and Jenness, T., “LSST Data Management Subsystem Requirements,” (July 2018).
- [5] Ivezić, Ž. and The LSST Science Collaboration, “LSST Science Requirements Document,” (Jan. 2018).
- [6] Sick, J., “The LSST the Docs Platform for Continuous Documentation Delivery,” (July 2016).
- [7] Dubois-Felsmann, G., “Camera Data Acquisition Interface,” (June 2015).

APPENDIX B. ACRONYMS

Acronym	Description
API	Application Programming Interface
ATM	Adaptavist Test Management
CI	Continuous Integration
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)
LaTeX	(Leslie) Lamport TeX (document markup language and document preparation system)
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