Documentation automation for the Verification and Validation of Rubin Observatory Software

Gabriele Comoretto[a], Leanne P. Guy[a], William O'Mullane[a], Keith Bechtol[b], Jeffrey L. Carlin[a], Brian Van Klaverer[c], and Austin Roberts[a]

[a]AURA/Rubin Observatory, 950 N. Cherry Ave., Tucson, AZ 85719, USA
[b]Department of Physics, University of Wisconsin-Madison, Madison, WI 53706, USA
[c]SLAC National Accelerator Laboratory, Menlo Park, CA, USA

March 12, 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 used in the preparation of documentation, for a systematic verification and validation documentation approach.

By adopting a systematic approach, we guarantee full traceability to system requirements, integration with the project's System Engineering model, and substantially reduce the time required for the whole process.

1. INTRODUCTION

The Rubin Observatory Data Management System (DM), as described in[1] , is responsible for creating the software, services, and systems that will be used to produce the observatory's 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 LSST images and to enable value-added contributions from the broader scientific community. 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 in[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, that maintains a history of the process. The System Engineering model is synced with the Jira test framework, providing a direct link between tests and requirements. Test documents can therefore be generated programmatically, 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 System Engineering model, see[3] , ensures full traceability to system requirements.

2. THE DOCUMENTATION PROCESS

In the scope of Verification and Validation, documents are written using LaTeXand considered as source code. Each document has a Git repository, where each edit is driven by a Jira issue, implemented in a ticket branch, reviewed by a third person before merging to master.

Using the GitHub **Pull Request**, it is easy to drive formal reviews, keeping the right contributors informed, and ensuring that changes are agreed.

Docsteady has been developed in-house by LSST DM to address this problem. Expand on this and explain why we developed this.

3. THE VERIFICATION AND VALIDATION

The verification and validation approach as illustrated in[3] , has been implemented. Required tools have been put in place, like *Syndeia* and *Docsteady*.

3.1 Tools

Follows the description of the tools we are using, of the procedure followed and the automatic generation of the test documents.

GitHub : It host the document content, and consider it as source code. It also provides metadata, like date, revision numbers and tags.

Travis : It is the continuous integration tool integrated with GitHub. In the case of documents, Travis is configured in such a way that, when a change in the repository happens, it triggers the build of the pdf and publish it to the corresponding lsst.io page.

lsst.io : Each document has a lsst.io publication page, where the pdf is pushed by Travis, after each successful build. This makes very easy to share the documents in different version, like for example tags or branches.

Docushare : It is the official Rubin Observatory documentation repository during construction. Documents are uploaded in Docushare only after their formal approval.

Jira : The Rubin Observatory issue tracking system. The Adaptavist Test Manager plugin (ATM) provides additional functionalities to Jira to manage test activities.

DocSteady : It is the tool that permits the generation of the test documents, extracting the information from Jira using REST API.

MagicDraw : It is the Rubin Observatory Modeling tool. It is used for requirement management. Verification Elements are created in MagicDraw and then synchronized to and from Jira.

Syndeia It is the tool that permits the integration between MagicDraw and Jira. The Verification Elements, first created in MagicDraw, are synched in Jira. After the test activity is completed, the information will be synched back to MagicDraw.

3.2 Procedures

The Verification and Validation activities originate from the requirements. We assume in this document, that the requirements have been properly formalized, documented and approved.

For each requirement in MagicDraw, one or more verification elements are created. When first created, the verification elements have no description. The only information they have is the requirement from where they have been generated.

They are synched to Jira using Syndeia, where they can be assigned to a responsible and completed with the relevant information.

[add picture?]

3.2.1 Test Procedures Preparation

After the verification elements have been created and propagated to Jira, they are assigned depending on the team or component they belong to. The assignee has the task to provide the right scope of the verification element and create the corresponding test cases. The verification elements are addressing all the aspects of a requirement, that need to be verified. Verification Elements are organized per component, like for example *DM*, and sub-component, like for example *Network*.

The test case associated with a verification element are assigned to an **owner**, that will complete it giving description, scope and, test procedure and other relevant information depending on the cases.

Verification elements and test cases are baselined in corresponding documents, in order to guarantee traceability and correctness of them.

3.2.2 Planning and Execution

Test campaigns are scheduled following defined milestones at project level, that are outside the scope of this document. Extra test campaigns may be scheduled on a per need base.

For each test campaign, two Jira ATM objects have to be created at least:

- **Jira ATM Test Plan** that provides the context of the test activity, and usually corresponds to a milestone.
- **Jira AATM Test Cycle**(s) that provides the scope. For each test campaign we may have multiple test cycles, depending on the different configurations, datasets, or extra conditions we may want to test. The Test Cycles are traced to the Test Plan, and provides the list of test cases that need to be executed.

We can identify two phase:

Planning : It is the phase when the test campaign is prepared. All relevant information shall be collected in the Jira ATM Test Plan and Test Cycle(s). At the end of this phase we shall be able to say: **the test is ready to start**. Despite in the Rubin Observatory there are no formal Test Readiness Review for each test campaign, the tooling and procedures in place permit to who is reposnsible of the test activity and relevant stakeholders, to assess and review the collected information. This is done extracting the information into a document, the *Test Plan*, in GitHub, generating the pdf and making it available in the corresponding lsst.io landing page. Contributors, reviewers and stakeholders can access easily the produced pdf, check the content of the test plan and comment, ask for clarification, or changes, using the GitHub Pull Request (PR) mechanism or the corresponding Jira issue. When agreement on the test plan content has been reached, the ATM 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 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. The Github repository will be tagged with the corresponding issue number *v1.0*.

Execution : In this phase, the testers identified in the the test plan are in charge to execute the test procedures and document the result of each steps in Jira ATM. The ATM plugin provide a test player view, where for each step in each test case, it is possible to say, it it has been executed successfully or not. Also it possible to related to the test execution any Jira issue documenting problems raised during the execution.

All this information is extracted in a report document. In order to avoid the proliferation of documents, the report information is added to the *Test Plan* created in the previous fare. The document is therefore renamed as **Test Plan and Report**.

Once the test execution is completed, an overall assessment shall be provided in the ATM Test Plan.

The document generation will provide a readable version of the document including all test information. Stakeholder are able to to review the outcome of the test campaign using the same PR mechanism reported above, commenting, asking for more information or changes if required, and finally, when consensus is reached, approve the test campaign result.

At the end of the test campaign, the Test Plan and Report is issued and uploaded to Docushare. The Github repository is tagged with the new issue number, *v2.0*.

3.3 Test Documents

Based on the process just outlined above, the following documents are identified.

These documents are generated using the **Docsteady**. The generation can be done manually, or automatically. Authomation is particularly useful in case we want to see every day the progress of the previous day published in the lsst.io landing page.

The extraction tool, Docsteady, permits to the user to concentrate only on the test activities, forgetting any documentation aspect.

3.3.1 Test Specification

The Test Specification is a document that baseline the test cases defined on a specific component.

3.3.2 Test Plan and Report

The Test Plan and Report include all planning and execution information. The document is issued in two times. The first issue corresponds to the consolidation of the planning activity, the second to the finalized test campaign.

3.3.3 Verification Elements Baseline

This document provides a snapshot of the verification elements, which content is maintained in Jira issues, therefore it is very easy to change. Having a snapshot of the verification elements in a document, make it possible to assess them, approve and keep history. Approved versions are uploaded in Docushare and used as a reference in other test documents.

4. INFORMATION SUMMARY AND THE VERIFICATION CONTROL DOCUMENT

As it has been described in the above section, all Rubin Observatory test information is managed in Jira. This permits to extract very easily the Verification Control Document (VCD), that shows the level of coverage for each requirement.

The VCD is a latex document generated using the document generation tool Docsteady. It is managed in the same way as the Test Plan and Report, the latex code is managed in Github, built in Travis and published in lsst.io.

Since the number of requirement for the Rubin Observatory is very high, the VCD is generated per subsystem.

The VCD provides 2 main sections:

- **Summary Information** where a status overview is given. This includes the number of requirements and verification elements related to passed or failed test cases.

- **Detailed Information** where for each requirement it is shown which are the verification elements and test cases and the status of the test cases. Links to the test documents are provided, but not descriptive information.

The VCD become an important document for management to know the level of verification and validation that has been achieved so far. At the same time it can be provided to reviews in order to demonstrate that the expected milestones have been met.

5. CONCLUSIONS AND OUTLOOK

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., Daues, 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).

APPENDIX B. ACRONYMS

Acronym	**Description**
API	Application Programming Interface
ATM	Adaptavist Test Management
AURA	Association of Universities for Research in Astronomy
DM	Data Management
DPAC	Data Processing and Analysis Consortium (Gaia)
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)
VCD	Verification Control Document