

NRAP Phase II   
Software Quality Assurance Plan

Version 1.1

February 21, 2018

Prepared by:

Timothy Jones

MATRIC

430 Drummond St. Suite 2

Morgantown, WV 26505

(800)611-2296 x905

[timothy.jones@matricresearch.com](mailto:timothy.jones@matricresearch.com)

**Disclaimer**

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference therein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed therein do not necessarily state or reflect those of the United States Government or any agency thereof.

**Acknowledgments**

This work was completed as part of the National Risk Assessment Partnership (NRAP) project. Support for this project came from the U.S. Department of Energy’s (DOE) Office of Fossil Energy’s Crosscutting Research program. The authors wish to acknowledge Traci Rodosta (Carbon Storage Technology Manager), Kanwal Mahajan (Carbon Storage Division Director), M. Kylee Rice (Carbon Storage Division Project Manager), Mark Ackiewicz (Division of CCS Research Program Manager), Darin Damiani (Carbon Storage Program Manager), Robert Romanosky (NETL Crosscutting Research, Office of Strategic Planning), and Regis Conrad (DOE Office of Fossil Energy) for programmatic guidance, direction, and support. NRAP research team also acknowledges the substantial effort put forth by the many internal and external reviewers from the CCS community, who through their many hours of tool testing and quality feedback, helped to improve the quality of NRAP’s tools

Table of Contents

Table of Contents i

List of Figures i

List of Tables i

Revision History ii

1. Introduction 1

1.1 Purpose 1

1.2 References 1

1.3 Overview 1

2. Organization and Responsibility Structure 1

2.1 Responsibility Structure Overview 1

2.2 Software Development & Documentation Team Responsibilities 2

2.3 Quality Assurance Team Responsibilities 3

2.4 Technical Writing/Documentation Team Responsibilities 3

2.5 Project Management Team Responsibilities 4

3. Software Development Life-Cycle Process 4

3.1 Software Development Life-Cycle Overview 4

3.2 Agile Development Approach 7

3.3 Milestone Traceability Methods for NRAP Software Development 8

3.4 Software Development Workflow for NRAP Tool Software Development 9

4. Software Testing Procedures 14

4.1 Software Testing and Procedures Overview 14

4.2 Types of Testing Performed 15

5. Software Configuration Management Process 16

5.1 Software Configuration Management Overview 16

5.2 Software Development Process using GitLab 17

Appendix A: Testing Definitions A-1

Appendix B: Acronym Definitions A-3

List of Figures

[Figure 1: NRAP Responsibility Structure 2](file:///C:\Users\timothy.jones\Desktop\NRAP%20Software%20Quality%20Assurance%20Plan%20revision%2020180220.docx#_Toc506886568)

[Figure 2: NRAP Software Development Life Cycle 5](#_Toc506886569)

[Figure 3: NRAP Application Lifecycle Management 6](file:///C:\Users\timothy.jones\Desktop\NRAP%20Software%20Quality%20Assurance%20Plan%20revision%2020180220.docx#_Toc506886570)

[Figure 4: Traceability Relationship Using GitLab 9](file:///C:\Users\timothy.jones\Desktop\NRAP%20Software%20Quality%20Assurance%20Plan%20revision%2020180220.docx#_Toc506886571)

[Figure 5: NRAP Software Development Workflow Using GitLab 10](#_Toc506886572)

[Figure 6: GitLab Issue Item Example 18](file:///C:\Users\timothy.jones\Desktop\NRAP%20Software%20Quality%20Assurance%20Plan%20revision%2020180220.docx#_Toc506886573)

[Figure 7: GitLab Notification Options 19](file:///C:\Users\timothy.jones\Desktop\NRAP%20Software%20Quality%20Assurance%20Plan%20revision%2020180220.docx#_Toc506886574)

[Figure 8: GitLab Burndown Chart Example 20](file:///C:\Users\timothy.jones\Desktop\NRAP%20Software%20Quality%20Assurance%20Plan%20revision%2020180220.docx#_Toc506886575)

List of Tables

Table 1: GitLab Issue Field Descriptions 18

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Timothy Jones | 09/29/17 | Initial Draft | 1.0 |
| Timothy Jones | 02/21/18 | Revised all sections to adjust focus on the cloud-based GitLab approach rather than JIRA. Simplified task workflow. | 1.1 |
|  |  |  |  |

# Introduction

The following three subsections introduce the purpose and overview of NRAP’s Software Quality Assurance Plan, including the references used throughout the document.

## Purpose

The Software Quality Assurance Plan describes the software development policies, procedures, and guidelines required of tools developed for the National Risk Assessment Partnership. Each development team is expected to adhere to the guidelines set forth in this document.

## References

The following references were utilized in the development of this document.

* GitLab Issues User’s Guide, *GitLab Inc.,* https://docs.gitlab.com/ce/user/project/issues/; *2018*
* GitLab Workflow Documentation, *GitLab Inc.,* https://docs.gitlab.com/ce/workflow/README.html; *2018*
* Software Testing and Quality Assurance, *SoftwareQATest.com, http://www.softwareqatest.com/; 2017*

## Overview

The remainder of this document consists of four additional sections and an appendix.

* Section two provides a description of the organization and responsibility structure.
* The third section provides a description of the software development life-cycle process and GitLab workflow.
* Section four provides a summary software testing performed.
* The fifth section discusses the software configuration management process using GitLab.
* Finally, the appendices include testing definitions that are used throughout this document.The appendix is a reference section to increase understanding of the content within the document.

# Organization and Responsibility Structure

The following five subsections explain the responsibility structure overview, high level responsibilities of the software development team, quality assurance team, technical writing/documentation team, and project management team.

## Responsibility Structure Overview

The following diagram displays the responsibility hierarchy.

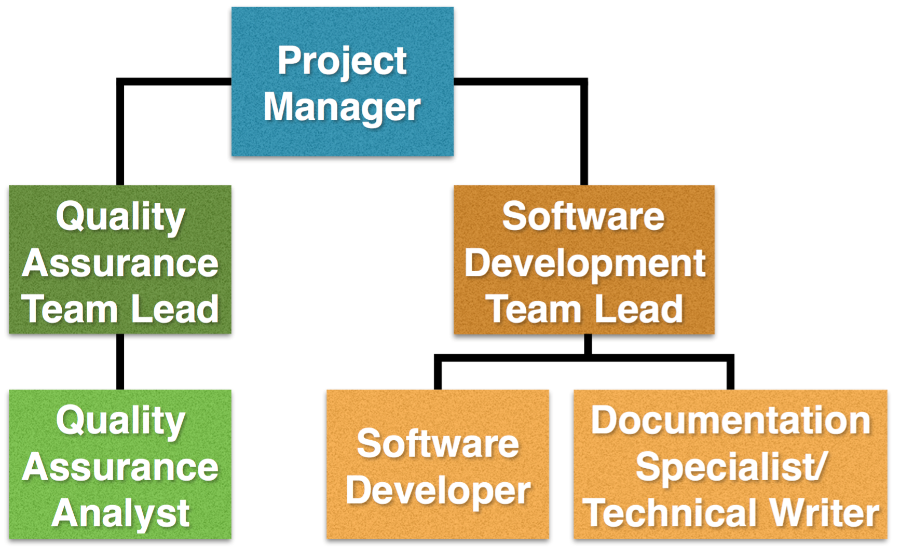


Figure : NRAP Responsibility Structure

The Project Manager oversees all operations for a specific project. The Quality Assurance Team Lead oversees the Quality Assurance Analysts assigned to the project. The Software Development Team Lead oversees Software Developers and Documentation Specialists assigned to the project. A person may be part of multiple roles in this responsibility structure.

## Software Development & Documentation Team Responsibilities

Lead Software Developer

* + Oversees all developer staff and activities for all assigned applications
  + Provides feedback to development team
  + Serves as an escalation point for issues involving software development
  + Serves as a resource and communication point for decisions about development schedule changes
* Develops software solutions by studying information needs; conferring with users; studying systems flow, data usage, and work processes; investigating problem areas; following the software development lifecycle
* Finds solutions to software development errors / debugging code
* Performs impromptu software testing including but not limited to unit testing, white box testing, and simple sanity testing
  + Provides status report of development progress during scrum meetings (if scrum is applicable to the team size)
  + Keeps up to date with new technologies, software, and coding methods
  + Ensure software development team members are obeying standard operating procedures regarding but not limited to: code commenting, source code tracking, and documentation.

Software Developer

* Develops software solutions by studying information needs; conferring with users; studying systems flow, data usage, and work processes; investigating problem areas; following the software development lifecycle.
* Finds solutions to software development errors / debugging code.
* Performs impromptu software testing including but not limited to unit testing, white box testing, and simple sanity testing
* Provides feedback to development team
* Provides Lead Software Developer with input for scrum meetings (if scrum is applicable to the team size)
* Keeps up to date with new technologies, software, and coding methods.

## Quality Assurance Team Responsibilities

QA Lead

* + Software Related Duties:
  + Oversees all testing staff and activities for all assigned applications
  + Ensures all testing deadlines are being met
  + Performs informal preliminary usability and functional testing on issues as they are completed
  + Organizes, oversees, and performs testing
  + Provides feedback to developers regarding test results
  + Serves as an escalation point for issues involving testing
  + Serves as a resource and communication point for decisions about test schedule changes
  + Provides status report of testing progress during scrum meetings (if scrum is applicable to the team size)
* Processes and Procedure Duties:
  + Oversees and delegates quality assurance validation duties to QA Analysts.

QA Analyst

* + Software Related Duties:
  + Performs informal preliminary testing on software tools as they are made available to the QA team for testing.
  + Performs usability and functional testing
  + Provides feedback to developers regarding test results
  + Provides Test Lead with input for scrum meetings (if scrum is applicable to the team size)
* Processes and Procedure Duties:
  + Performs quality assurance usability and functional tests.

## Technical Writing/Documentation Team Responsibilities

Technical Writer/Documentation Specialist

* Develops, writes, and maintains information for internal and external audiences required under contract such as installation guides, user manuals, quick start manuals, and other documentation. Within the scope of the NRAP projects, the technical writers’ focus is to create a common structure with the NRAP user manuals, adding standardized language to meet documentation requirements, and formatting the content provided by the software development teams into the uniform NRAP user manual structure.
* Provides status updates on projects as required by management. Coordinates with other technical writers on documentation projects.
* Performs technical and peer reviews of product documentation.

## Project Management Team Responsibilities

Project Manager

* + Oversees all staff and activities for all assigned applications.
* Meets financial objectives by forecasting requirements; preparing budgets; scheduling expenditures; analyzing variances; and initiating corrective actions.
* Responsible for communication, including status reporting, risk management, escalation of issues, and making sure the project is delivered in budget, on schedule, and within scope.

# Software Development Life-Cycle Process

The following four subsections explain the overview of NRAP’s software development life-cycle process, agile development approach, traceability methods, and software development workflow for projects.

## Software Development Life-Cycle Overview

NRAP’s Software Development Life-Cycle (SDLC) is intended to drive the development of software systems in a very deliberate and structured manner. The SDLC is vital for NRAP’s software developers, QA analysts - testers, documentation specialists, and project managers. This SDLC foundation is coupled with agile development methods for an iterative means of managing and organizing projects. NRAP’s agile methods are explained in further detail in section 3.2.

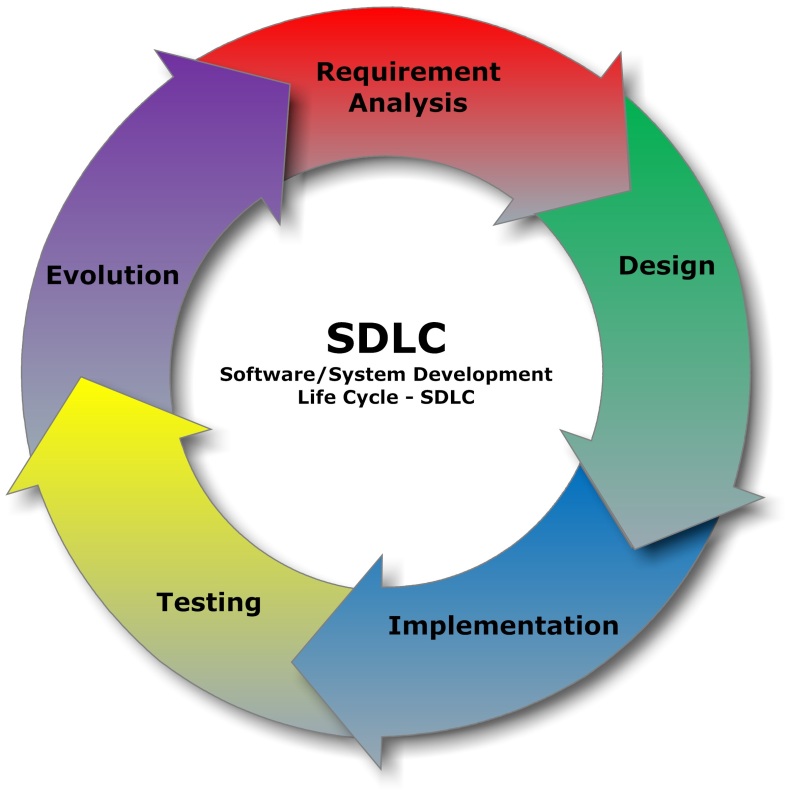


Figure 2: NRAP Software Development Life Cycle

* **Requirements Gathering, Investigation, and** **Analysis -** The 1st stage of SDLC is the Requirements gathering phase. During this stage, information technology solutions are discussed. Multiple alternative projects may be suggested and their feasibility analyzed. Operational feasibility is assessed, and it is determined whether or not a project can meet end user needs. In addition, an economic feasibility investigation is conducted to judge the costs and benefits of the project. Technical feasibility must also be analyzed to determine if the available hardware and software resources are sufficient to meet expected specifications. If it is decided to move ahead, a proposal should be produced that outlines the general specifications of the project.
* **System Analysis and Design** - The goal ofsystem analysis and design is to determine the best approach to solve a specific set of problems. This step involvesbreaking downthe system in different pieces to analyze the situation, analyzing project goals, breaking down what needs to be created and attempting to engage users so that definite requirements can be defined.   
  Design elements describe the desired software features in detail, and generally may include functional hierarchy diagrams, screen layout diagrams, process diagrams, pseudo-code, an entity-relationship diagram, and data dictionary.
* **Implementation and Development** - Implementation and development operations are described in detail, including screen layouts, business rules, process diagrams and other documentation. The development stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced. Software developers will engineer and write/code the software to accomplish all of the requirements that were set forth by the requirements gathering process.
* **Testing** - The code is tested at various levels; unit, system, user acceptance, and other forms of testing are performed. See section 6 for NRAP specific testing procedures and information.
* **Evolution of Operations and Maintenance** - Thedeploymentof the system includes changes and enhancements.Maintainingthe system is an important aspect of SDLC. As key personnel change positions in the organization, new changes may be implemented, which will require standard procedures for maintenance and operations on a case-by-case basis determined by the contract.

NRAP’s Application Lifecycle Management (ALM) consists of six parts, project management, requirements management, design & architecture, development, QA & testing, release management, and optimizations.

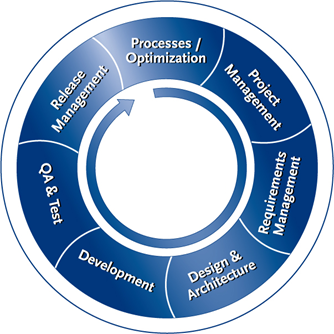


Figure : NRAP Application Lifecycle Management

* **Project management** oversees the project ensuring that all contractual obligations and project objectives are met while honoring the preconceived constraints. These constraints consist of the project’s scope, completion deadlines, quality and budget.
* **Requirements management** is the process of analyzing, documenting, tracing, and prioritizing the system requirements. The purpose of requirements management is to ensure that NRAP documents, verifies, and meets the needs and expectations of its users and internal or external stakeholders.
* **Design & architecture** is the process of interpreting customer requirements, implementing a plan for software architecture, defining how the application will carry out the tasks and what software technologies and tools are to be used for a successful outcome of the software project. Hence, the requirements are represented in a comprehensible way for software developers to implement a software solution.
* **Development** is the process of engineering and writing/coding the software. The software is designed and produced to accomplish all of the requirements that were set forth by the requirements gathering process.
* **QA & testing** involves the efforts of software testers and developers. Testing is often closely related with other stages of the software development lifecycle such as development and design & architecture. Quality assurance verification involves performing a variety of testing procedures and checking compliance to the contractual requirements.
* **Release management** encompasses the identification, packaging, and delivery of the software product. This is including but not limited to the software, documentation, release notes, and configuration data.
* **Optimization** is the process of modifying an existing software system to use fewer resources or make some aspect of it work more efficiently. Optimizations are made to improve the functionality of the software.

All six ALM processes are continuous throughout the life of the project

## Agile Development Approach

NRAP’s tool development teams follow an agile development approach.

In agile software development, work is confined to a regular, repeatable work cycle, known as a sprint or iteration. Iterations may be adjusted based on the time frame for deliverable deadlines, etc. During each sprint, the team creates finished portions of the software product. The set of features that go into a sprint come from the product backlog, which is an ordered list of requirements. Which backlog items go into the sprint (the sprint goals) is determined during the sprint planning meeting at the beginning of an iteration or sprint. During this meeting, the team is informed of the items in the product backlog that they need to complete (typically the ones with the highest priority). The team then determines how much of this they can commit to complete during the new iteration or sprint, and records this in the sprint backlog. The sprint backlog is maintained in NRAP’s GitLab solution. GitLab is an issue tracking and source code repository system by GitLab Inc.

Source code developed and maintained by each developer must be checked in to GitLab on a regular basis. GitLab is a web-based service used for source code and development projects that use Git revision control. Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows.

Frequent code commits are a healthy routine for the software development life cycle and tracking the lifespan of each issue. NRAP requires the development team to include two key components for traceability when committing code into the GitLab repository:

1) A thorough comment for the changes made in the source code for the commit about to take place. 2) A link or related description to the issue(s) being addressed in the commit about to take place.

Meetings may be planned throughout an iteration (sprint) to discuss the backlog of requirements, code review sessions, or other review meetings.

A code review session consists of the majority of the developers on the project getting together to review each line of code to discuss its functionality and add additional descriptive comments into the body of the source code. Developers will also share code improvement ideas to potentially increase the efficiency of the existing code. At least one code review should occur every six months or prior to a major product release to the public.

## Milestone Traceability Methods for NRAP Software Development

Milestone traceability refers to documenting the life of a milestone and its relationship with various issue items associated within a GitLab project. GitLab’s issue item can describe one or many tasks or bugs that must be completed or resolved in order to accomplish a milestone. Being able to trace each issue item back to a milestone is vital for the organization and management of a project. Milestones come from different sources such as the contract/statement of work, end-users, developers, etc. Depending on the level of effort or priority, internally developed milestones are simply added as an issue item with a prioritization level known as a “weight” that may not always map to a milestone.

NRAP tool development teams use GitLab to organize and manage all issue items, milestones, and source code. NRAP tool development teams follows the standard agile GitLab workflow as a basis for issue item lifespan management. The default GitLab lifespan for an issue consist of three states (in order): To Do, Doing, and Closed. The following image depicts the relationship of a milestone and an issue item used by NRAP:

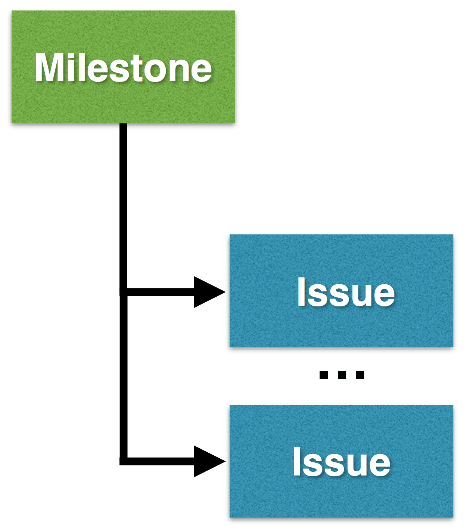


Figure : Traceability Relationship Using GitLab

A milestone must have at least one issue associated with it before the milestone may be closed (completed).

## Software Development Workflow for NRAP Tool Software Development

The diagram below defines the workflow used in NRAP’s traceable software development process using GitLab:

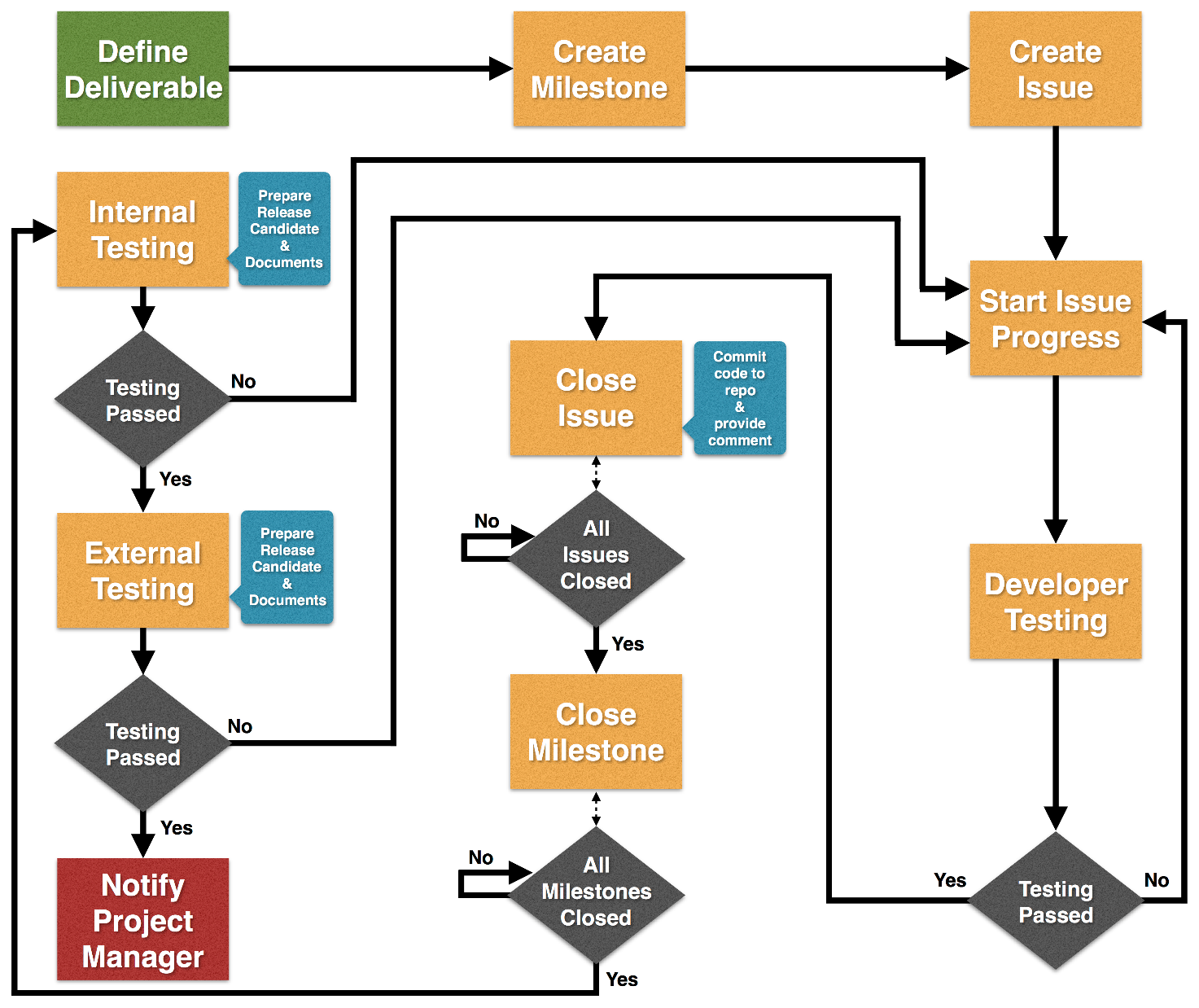
****

Figure : NRAP Software Development Workflow Using GitLab

1. **Define deliverables with NRAP management**
   * Define a list of requirements and deliverables with the NRAP management team.
   * Obtain feedback from managers or other team members to be added as an additional milestone or issue.
2. **Create milestone that addresses deliverable**
   * Creating a milestone is completed by the software development team leader or a developer given permission to do so by the management team or software development team leader. Create a milestone in GitLab that completes a portion of a deliverable.
   * **Adding Milestone:**
     + Using GitLab, the software development team lead will create a milestone by following the steps in the “GitLab Workflow Documentation” reference in section 1.2 to create milestones in GitLab.
     + The new task must contain the following details before being created:
       - **A descriptive title**
       - **A description of the work item; explaining what needs accomplished**
       - **A start date that defines when progress will begin to complete the milestone**
       - **An end date that defines the estimated time until completion or deadline of the milestone**
3. **Create issue and associate with existing milestone**
   * Creating an issue is completed by the software development team leader or a software developer. Create an issue in GitLab that completes a portion of a milestone and assign it to one of the software developers.
   * **Adding Issue:**
     + Using GitLab, the software development team lead or software developer will create an issue by following the steps in the “GitLab Workflow Documentation” reference in section 1.2 to create issue items in GitLab.
     + The new task must contain the following details before being created:
       - **A descriptive title**
       - **A description of the work item; explaining what needs accomplished**
       - **Assignment of the work item to a team member**
       - **Assignment of a milestone item that the issue is addressing**
4. **Start progress on issue item**
   * Changing an issue in GitLab from “To Do” state to a “Doing” state is completed by the software developer assigned to the work. When a developer changes the issue’s status to “Doing”, it signifies that the developer has focused their work efforts on that issue. When an issue is deemed “Doing” status, an optional email alert is sent out to the development team leader if they have opted in for notifications specifically for label updates or all updates with items changed within the project.
   * **Updating Issue Status to “Doing” State:**
     + Using GitLab’s web portal, the software developer will change the status of an issue by first locating it using the issue board. Once located, drag the issue item from the “To Do” list into the “Doing” list. This will update the label and status. The developer can officially begin the work described in the task.
5. **Perform developer testing**
   * Developer testing is performed by a software developer or software development team lead. Once an issue item has been developed by a software developer, the issue will be tested using unit tests that are developed to specifically validate the components created to address the issue.
     + If the developer testing fails, the developer will address any outstanding issues with the code and retest.
     + If the developer testing passes, software developer will close the issue.
6. **Close issue item**
   * Closing an issue in GitLab is completed by the software development team lead or the developer who had completed the work described in the issue item. An issue is closed when it has passed developer testing. The issue item will be tested based on whether it passes unit tests or any other developer specific tests that are defined by the software development team lead.
   * **Closing Issue:**
     + In GitLab’s web portal, the software developer will change the status of an issue by first locating it using the issue board. Once located, drag the issue item from the “Doing” list into the “Closed” list. This will update the label and status. Optionally, the software developer is encouraged to include a note in the “Comment” section of the issue explaining a brief overview of the testing results or process. This can be completed by navigating to the issue’s overview page. Closing issues should only occur after the source code has been committed to the GitLab repository.
7. **Close milestone item**
   * Before closing a milestone item, the software development team lead ensures that all issues that are associated with the milestone have been closed. The software development team lead shall review all issues associated with the milestone and ensure that all issue items address the milestone’s objective. If part of the milestone is not addressed, create new issues and continue the development phase.
   * **Closing Milestone:**
     + In GitLab’s web portal, the software development team lead will change the status of a milestone by first locating it using the milestone dashboard. Once located, select the “Close Milestone” option next to the milestone that you wish to close. This will update the status.
8. **Internal Testing**
   * Prior to the start of internal testing, the software development team in coordination with the documentation specialists, shall generate a draft of a user manual that will accompany the software release candidate when it is distributed for testing.
   * The internal testing release candidate will be distributed to NETL’s Energy Data Exchange (EDX) as a private submission within a private collaborative workspace specifically for testing NRAP software products and may also be distributed to other approved locations determined by NRAP project managers. Testers will be added to the private workspace on EDX so they may securely download the software for testing.
   * Internal testing is performed by at least two other partner lab teams that have little to no direct involvement with the NRAP software that was developed. Internal testing includes basic usability and functional testing. Testers will be interacting with the software’s user interface and performing standard functions that the software was designed to compute. Each partner lab team is responsible for providing feedback to the software development team lead associated with the project. This feedback shall include but is not limited to: bugs found, improvement/enhancement requests, or change requests to an existing feature. New issues and milestones may be created to address feedback provided in the internal testing phase.
   * Project managers and software development team leads will decide what items shall be addressed based on the feedback provided by the external collaborators.  
       
     Outcomes of internal testing:
     + Failing the test – if this occurs, the issue shall be re-opened by the software development team lead and the software developer who was originally assigned to that issue shall be notified.
     + Passing the test – if this occurs, the software release candidate is ready to proceed with external testing, pending any modifications that the software development team may need to perform to ensure the safety of its intellectual property prior to external release.
9. **External Testing**
   * The external testing release candidate will be distributed to NETL’s Energy Data Exchange (EDX) as a private submission within a private collaborative workspace specifically for testing NRAP software products and may also be distributed to other approved locations determined by NRAP project managers. Testers will be added to the private workspace on EDX so they may securely download the software for testing.
   * External testing is performed by NRAP approved third party collaborators that are not part of the NRAP multi-national lab team. External testing includes basic usability and functional testing. Testers will be interacting with the software’s user interface and performing standard functions that the software was designed to compute. Each external testing team is responsible for providing feedback to the software development team leads or project managers. This feedback shall include but is not limited to: bugs found, improvement/enhancement requests, or change requests to an existing feature. New issues and milestones may be created to address feedback provided in the external testing phase.
   * Project managers and software development team leads will decide what items shall be addressed based on the feedback provided by the external collaborators.

Outcomes of external testing:

* + - Failing the test – if this occurs, the issue shall be re-opened by the software development team lead and the software developer who was originally assigned to that issue shall be notified.
    - Passing the test – if this occurs, the software release candidate is ready for public release pending project manager approval and any modifications that the software development team may need to perform to ensure the safety of its intellectual property prior to public release.

1. **Notify project managers of completion of deliverables**
   * Once external testing has been completed and feedback vital to the successful operation of the software has been addressed, a final version of the user manual shall be created and bundled with a public release version of the developed software. Software development leads will notify the project managers of the completion of the deliverables once a finalized software bundle is completed and ready for distribution.
   * The public release version will be distributed to NETL’s Energy Data Exchange (EDX) as a public submission and may also be distributed to other approved locations determined by NRAP project managers.

# Software Testing Procedures

The following two subsections provides an overview of software testing and procedures and describes the types of testing performed.

## Software Testing and Procedures Overview

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. NRAP’s testing procedures have been established to ensure that the software being developed will meet all requirements. NRAP’s testing procedures validate and verify that the software/application/product:

1. Meets the system requirements as guided by the stakeholders and users
2. Meets the technical requirements as guided by its design and development
3. Works as expected

NRAP’s agile development approach employs test-driven development; this places a portion of the testing in the hands of the developer, before it reaches the community of testers. This approach allows for two stages of testing: the unit testing performed by the developers and usability and functional testing completed by both internal and external collaborators.

At least two internal collaborators from other labs that are part of NRAP that are not involved with the development of the product will perform basic usability and functional testing and provide feedback to the software development team leads. After the initial internal testing is completed a second phase of testing will occur with external collaborators that are selected by NRAP to perform basic usability and functional testing. The external collaborators will provide feedback on product to the software development team leads responsible.

Both internal and external testing phases will be use NETL’s Energy Data Exchange (EDX) as a means to distribute software release candidates. A private submission containing the software release candidate shall be compressed, uploaded and contributed to a private collaborative workspace specifically for testing NRAP software products. The software release candidate may also be distributed to other approved locations determined by NRAP project managers. Testers will be added to the private workspace on EDX so they may securely download the software for testing. Each project may choose to optionally track the download activity of their software product through EDX for further verification and validation purposes.

## Types of Testing Performed

NRAP tool developer teams perform a variety of testing over the course of each development iteration (or sprint). Testing is performed in for two stages, unit testing performed by the developers and usability and functional testing completed by both internal and external collaborators.

The first stage of testing is known as impromptu developer testing. This is testing occurs while the developer is actively working on a task or bug. Impromptu developer testing mainly focuses on the internal structure of the program opposed to the functionality exposed to the end user. Impromptu developer testing includes: unit testing, white box testing, and simple sanity testing.

The second stage of testing will include usability and functional testing. These tests occur once all issues related to a milestone for a given iteration (sprint) or release has been tested by the software developers. As mentioned in section 4.1, internal and external collaborators will perform tests and provide feedback to the development team who created the product. The software development lead, QA lead, or project management team may request additional unique tests to be performed during this stage on a case by case basis.

Specifically, for software development, verification and validation (V&V) is the process of checking that a software system meets specifications and that it fulfills its intended purpose. Two important questions must be answered regularly throughout the software development life cycle.

Validation: Are we building the right product?

Verification: Are we building the product right?

In other words, validation ensures that the product actually meets the user's needs, and that the specifications were correct in the first place, while verification is ensuring that the product has been built according to the requirements and design specifications.

Regarding internal verification and validation of development procedures, NRAP’s software development workflow shall ensure that a member of the team reviews the efforts of another via a simplified code review prior to the release of the product to the general public. Part of each NRAP tool development team’s duty is to ensure that NRAP’s standard operating procedures for source code commits, in-line comments, testing procedures, issue lifespan, and traceability are being followed at all times. The issues and comments are reviewed by the NRAP team at the end of each software build iteration to ensure clarity of what was completed.

At the end of a project, additional testing will occur such as installation testing to ensure that the newly deployed system installs correctly.

See Appendix A for testing definitions.

# Software Configuration Management Process

The following two subsections provide a software configuration management process overview and NRAP’s use of GitLab for software configuration management for NRAP tool development efforts.

## Software Configuration Management Overview

Software conﬁguration management (SCM) is the discipline of controlling the evolution of complex software systems. It improves productivity by reducing or eliminating potential confusion caused by interaction among multiple team members on a project. Software configuration management is more than just the version control software used to organize a software development project. It is the process that defines how to control and manage change of the project.

All the members of the project are involved in the Software Configuration Management activities:

* Software developers implement software enhancements and report, analyze and fix defects.
* The team may use GitLab to retrieve a particular baseline, to report defects, or to propose software enhancements.
* Quality Analyst - testers report defects and document them in GitLab.

The following list is a high level overview of NRAP’s software configuration management tasks:

* Software Build Management – Managing the tools and processes utilized for software builds and versions.
* Process Management – Ensuring adherence to NRAP’s standard operating procedures for software development.
* Environment Management – Managing the hardware and software that hosts the system being developed.
* Team Management – Managing the team’s interaction relating to each sprint deadline and the project as a whole.
* Configuration Identification – Identifying configuration items, baselines, issue completion timeframe, release notes, and any additional information needing to be identified on a case by case basis.
* Configuration Auditing – Ensuring that configurations contain all of their intended parts based on documentation, milestones, and architectural specifications.
* Configuration Control – Implementing a controlled change process. The Lead QA Analyst, Lead Software Developer, and Project Manager may have change control meetings when change request(s) arise. These meetings are to approve or reject all pending change requests that are set against any baseline issue item existing in the GitLab Issues tracking system.

All baselines, configurations, trunks, branches, and release candidates are managed through GitLab. A new branch is developed at each iteration (sprint) and is merged into the code trunk once all developer testing has been completed and project management approval has been obtained, if needed. A release candidate branch is created when a stable release is ready for deployment.

## Software Development Process using GitLab

NRAP uses GitLab Issues for data collection, project tracking, milestone traceability, and reporting; NRAP also uses GitLab to manage source code control.

GitLab Issues, the task tracking component of GitLab, revolves around the issue. An issue is a single unit of work which needs to be completed. A GitLab issue item contains fields to define its objective, a few examples include: title, description, weight, assignee, milestone relationship, label, due date, file attachments, and few other attributes.

Below is an example of a GitLab issue item displayed in a web browser.

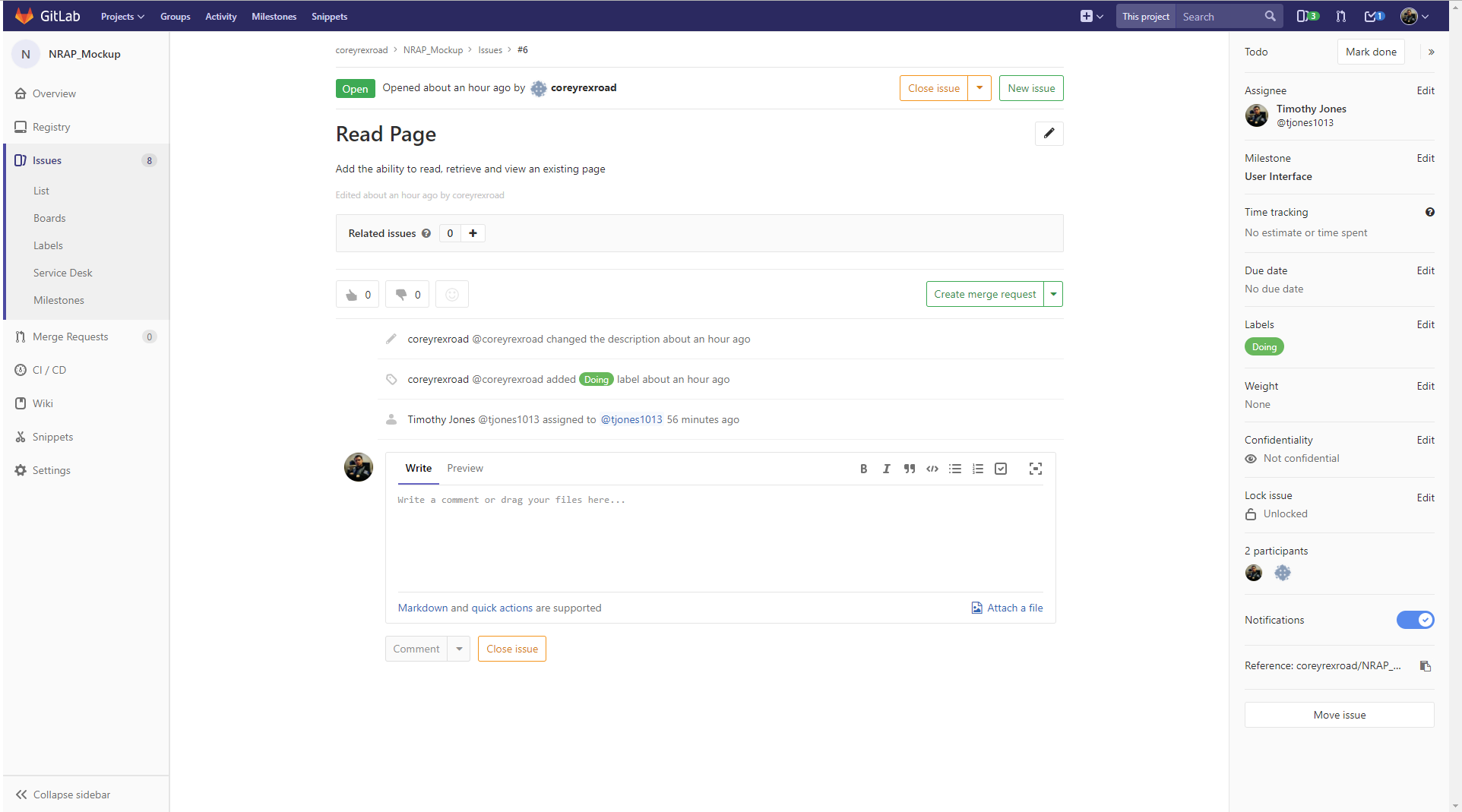


Figure : GitLab Issue Item Example

When creating a new GitLab issue item, at minimum, all users should complete the following fields: Title, Description, Label, and Assignee.

The following table describes relevant GitLab issue item fields:

|  |  |
| --- | --- |
| **Field** | **Description** |
| **Title** | The objective of the item summarized into a single sentence. |
| **Description** | Further explanation of the objective to be accomplished. |
| **Assignee** | Person assigned to perform the work on issue item. |
| **Milestone** | Deliverable/Requirement that the issue is associated with. |
| **Label(s)** | An identifier to describe a group of one or more specific issues (e.g. To Do, Closed). |
| **Weight** | The importance/priority of an issue in relation to other issues within the project. |
| **Due Date** | Date in which the issue must be completed and closed. |

Table : GitLab Issue Field Descriptions

NRAP follows the Standard GitLab workflow process template as a basis for issue item lifespan management. These items are internally stored and their schema can be customized to add other attributes to different items, or create new items on a per-project basis.

Developers may opt-in to receive notifications via the GitLab web portal and via email when the creation or a change has been made to an issue. These notifications provide a link to the detailed work completed in said issue item. These preferences can be modified under the User Settings > Notifications section of GitLab.

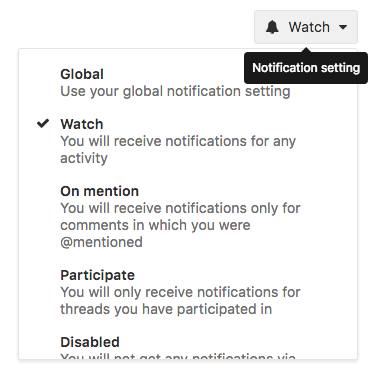


Figure : GitLab Notification Options

Basic reporting features within GitLab allow a variety of information to be generated including: lists of Backlog, To Do, Doing (in-progress), and Closed items, basic burndown rate per milestone. These reporting capabilities help project managers and developers gauge the status of progress throughout the software development lifecycle.

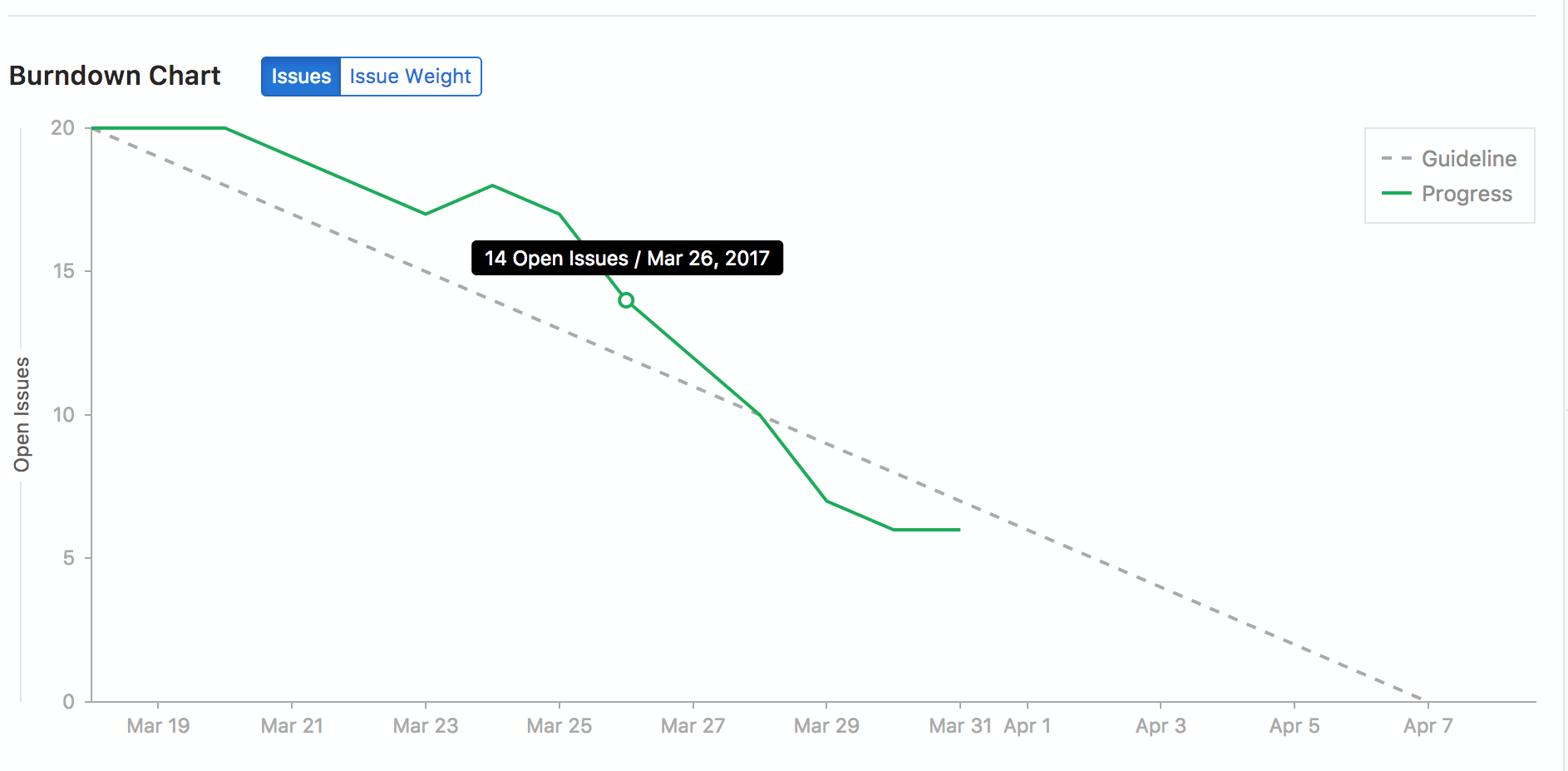


Figure : GitLab Burndown Chart Example

A burndown chart’s display can be based on the issues’ weights rather than the total count of issues currently open versus closed. This may provide better insight into the level of effort that remains due to the priorities and weights assigned to each issue item within the milestone.

Appendix A: Testing Definitions

***This appendix provides a list of different types of tests that may be performed during the testing phases of NRAP tool development.***

**Black-box testing** – **treats the software as a "black box", examining functionality without any knowledge of internal implementation. The tester is only aware of what the software is supposed to do, not how it does it.**

**Compatibility testing** – testing conducted on the application to evaluate the application's compatibility with the computing environment. Computing environment may contain some or all of the below mentioned elements:

* Computing capacity of Hardware Platform
* Bandwidth handling capacity of networking hardware
* Compatibility of peripherals (Printer, DVD drive, etc.)
* Operating systems (Linux, UNIX, Windows, Mac, etc.)
* Database (Oracle, Sybase, DB2, etc.)
* Other System Software (Web server, networking/ messaging tool, etc.)
* Browser compatibility (Firefox, Netscape, Internet Explorer, Safari, etc.)

**Functional testing** – This type of testing ignores the internal parts and focus on the output is as per the requirement. It is a black-box type testing geared towards functional requirements of an application.

**Installation testing** – testing to ensure that the newly deployed system is installed correctly and works properly on the customer’s hardware.

**Load testing** – testing to check system behavior under load; testing an application under heavy loads, such as testing of a web site under a range of loads to determine at what point the system’s response time degrades or fails.

**Performance testing** – testing performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

**Requirements Acceptance Testing** – The development team, QA team, and project management all agree that a specific user requirement set forth by the contract has been met.

**Sanity testing** – is a basic test to quickly evaluate whether a claim or the result of a calculation can possibly be true. It is a simple check to see if the produced material is rational (that the material's creator was thinking rationally, applying sanity). The point of a sanity test is to rule out obviously false results, not to catch every possible error.

**Security testing** – tests whether the system be penetrated by any hacking method. Testing how well the system protects against unauthorized internal or external access.

**Stress testing** – tests that determine the robustness of software by testing beyond the limits of normal operation. Stress testing is particularly important for "mission critical" software, but is used for all types of software. Stress tests commonly put a greater emphasis on robustness, availability, and error handling under a heavy load.

**System testing** – is testing conducted on a complete, integrated system to evaluate the system's compliance to its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

**Unit testing** – **refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors. These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.**

**Usability testing** – **testing that measures the usability, or ease of use, of a specific object or set of objects within the software. Usability testing aims to discover errors and areas that could be improved by simply interacting with the software as a user normally would. Usability testing is considered a black-box testing technique.**

**White-box testing** – **tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design tests. The tester (often times the developer) chooses inputs to exercise paths through the code and determine the appropriate outputs.**

Appendix B: Acronym Definitions

***This appendix provides a list of acronyms found throughout the document.***

|  |  |
| --- | --- |
| **ALM** | **Application Lifecycle Management** |
| **NRAP** | **National Risk Assessment Partnership** |
| **QA** | **Quality Assurance** |
| **SDLC** | **Software Development Life Cycle** |
| **V&V** | Verification and Validation |