DMSD Software Development Processes

PREPARED BY	**ORGANIZATION**	**DATE**
Rafael Hiriart	NRAO	6/6/2016
Morgan Griffith	NRAO	4/13/2018

APPROVALS (Name and Signature)	**ORGANIZATION**	**DATE**
Brian Glendenning	NRAO	4/13/2018
Approved by M. Griffith and B. Glendenning via NRAO Workflow.		

RELEASED BY (Name and Signature)	**ORGANIZATION**	**DATE**

Change Record

VERSION	**DATE**	**REASON**
0.1	01/18/2016	Draft version for review.
0.2	03/11/2016	Incorporating feedback from DMSD Leads, Eduardo Miranda and Lory Wingate.
0.3	06/06/2016	Incorporating a section about notifications, and introducing the roles of component lead and scientist.
0.4	02/16/2018	Fix reference links, M. Griffith to make edits
1.0	4/13/2018	Incorporating feedback from teams

VERSION	DATE	REASON

Table of Contents

1 Introduction

1.1 Purpose

This document describes the software development processes adopted by NRAO Data Management and Software Department.

1.2 Change Management

Changes to this document must be approved by the DMSD Software Division Head and subsequently distributed to all members of the DMSD Software Leads team.

1.3 Scope

The processes defined in this document are mostly based on the Recommendations for Improvement report for the CASA group [1]. The recommendations outlined in this report have been generalized so they can be adopted by other DMSD development groups besides CASA. See Section 2 for the list of recommendations, and details about the way they have been addressed or implemented.

The processes described in this document will be adopted by all development groups in DMSD, with three exceptions:

1. For now, it was decided that Green Bank development group will continue with its already established software development processes. This decision will be re-evaluated after the results of the GBT divestiture are known.
2. ALMA development groups in NRAO (Control/Correlator and Scheduling) will continue to follow ALMA development processes.
3. AIPS and Obit development groups will also continue with their current development processes.

ALMA development groups are bound to follow the already defined ALMA development processes. It is worth noting, however, that to a large extent the development processes defined in this document are compatible with ALMA processes, and the experience from ALMA NRAO development groups have been incorporated. See [2] for a description of ALMA software delivery process.

This document doesn't try to define all possible processes related with the software development activities. Instead, it defines a framework, prescribing some essential processes and artifacts, but leaving others to the discretion of each development group (and DMSD management), which should apply standard methods and practices from system and software engineering to the specific conditions of its development environment. The focus is on defining standard roles, responsibilities and interfaces between DMSD and its external stakeholders.

Software development processes will evolve as the organization matures and accumulates experience. This document should be viewed as a first step toward standardizing DMSD software development processes. Some aspects that have been left for each group to define and implement could be normalized across DMSD in future revisions.

1.4 Document Structure

The first part of this document discusses several general subjects. As mentioned before, Section 2 provides traceability between the process recommendations from [1] and the processes defined in this document. Section 3 defines software verification and validation, two terms that are used extensively in the rest of the text. Section 4 describes the way DMSD development processes should be applied in *Agile* projects. Section 5 discusses the subject of Release Management, while Section 6 deals with the topic of Software Configuration Management.

The second part of this document describes the DMSD software development processes in detail. The description is organized around requests, workflows, roles and artifacts. When a request to do something (fix a bug, implement a new feature, etc.) is submitted to a DMSD development group, its associated workflow is triggered. During the execution of a workflow, the process moves through several defined states. Explicit actions have been defined for each state, which need to be completed by a designated person that assumes a role (some role assignments are static, e.g. Project Scientist; while in other cases the assignee changes depending of each request, e.g., Reporter). Some of these actions involve the production of one or more artifacts, which normally need to be reviewed and accepted for the workflow to move to subsequent states.

The different request types are defined in Section 7, along with the process by which they are scheduled. Section 8 defines roles. Sections 9 to 13 define the workflows associated with each type of request. Finally, Section 15 provides templates for development artifacts mentioned in the rest of the text.

1.5 References

[1] Eduardo Miranda, "Recommendations for Improvement. Evaluation of CASA PMD Requirements RFP CV-740", 2015.
[2] Erich Schmid, George Kosugi, Jorge Ibsen, Morgan Griffith, "ALMA Software Delivery Process", COMP-70.05.00.00-0012-D-PLA, 2015.
[3] Jez Humble, David Farley, "Continuous Delivery", ISBN 978-0-321-60191-9, Addison-Wesley, 2011.
[4] Suzanne Robertson, James Robertson, "Mastering the Requirements Process, Third Edition", ISBN 978-0-321-81574-3, Addison-Wesley, .

1.6 Abbreviations and Acronyms

ALMA	Atacama Large Millimeter Array
CASA	Common Astronomy Software Applications
DMSD	Data Management and Software Division
JIRA	Atlassian (https://www.atlassian.com) software for issue and project tracking
NRAO	National Radio Astronomy Observatory
SCM	Software Configuration Management
SRS	Software Requirements Specification

2 CASA PMD Requirements Project Recommendations Traceability

This section documents how the recommendations prescribed as part of the "Evaluation of CASA PMD Requirements" project [1] have been addressed in this document. Note that this document attempts to define common software development processes for all DMSD groups, not only the CASA group or any other specific group. Some recommendations, which are only applicable to CASA, are not elaborated further. This doesn't mean they won't be implemented, it just means that they are not in the scope of this document, and should be elaborated and implemented by the CASA team. Similarly, other recommendations that are not directly related to development processes, but other organizational and management aspects of DMSD (team co-location, training), are not discussed further.

1. **Implement alternate periods of development and housekeeping.**

   Implemented, see Section 7.

2. **Strengthening the authority of the Project Scientist, Project Manager and System Architect.**

   See Section 8, which defines these roles and their respective responsibilities.

3. **Establish processes for work request management; work planning and tracking; configuration management; software development; verification and validation; building, deployment and releasing of software; quality assurance; and measurement and analysis.**

   This document provides guidelines and a framework for work request management, work planning and tracking, configuration management, and verification and validation. The rest are group-specific (i.e., recommendations should be evaluated and implemented by each group).

4. **Workload management policy.**

   All the changes recommended under this item are covered, with the exception of items 7 and 8. Item 7 (reserve one day per week for unplanned tasks) is left to the discretion of each group. Item 8 (expediting is restricted to the CASA User Committee) is CASA specific.

5. **Implement as soon as possible (ASAP) scheduling.**

   Covered in Section 7.

6. **Perform early estimation process.**

   Integrated in all the workflows. Performing an early estimation is done in the Unscheduled state. The T-shirt sizing technique is recommended.

7. **Maintain Work schedule and Resource availability databases.**

   This will be implemented in JIRA and Confluence.

8. **Improving quality of work requests.**

   All the proposed request types have been incorporated in this document (some names differ slightly). All the workflows incorporate an initial tollgate in the Open state, where the quality of the request is evaluated.

9. **Planning and tracking different types of work.**

   This document specifies specific workflows and templates for each type or request. Note that no distinction is made between the types of request and types of work. Both are mapped to the request types defined in this document. Besides this mapping, all of them are covered.

10. **Implement requirements traceability.**

    Not covered in this document. This is left for each group to evaluate and implement.

11. **Implement testing framework.**

    In general, it is covered in the workflows and the adoption of a continuous integration server. Implementation details are group-specific.

12. **Co-location.**

    Not covered in this document.

13. **Create (process) quality assurance role.**

    See Section 8, which defines this role and its responsibilities.

14. **Pay technical debt.**

    It is recommended that each group adopt housekeeping periods. Details about evaluating the technical debt and formulate plans to reduce it are left for each group to define.

15. **Implement coding guidelines.**

    For now, this is left for each group to define and implement.

16. **Create and maintain the CASA roadmap.**

    This is CASA specific, although each DMSD group will maintain a medium-term management plan (see Section 18).

17. **Training.**

    Not covered in this document.

3 Software Testing Verification and Validation

Software and systems engineering makes a distinction between software *verification* and *validation* testing. These terms are not interchangeable. For the purpose of clarity, they are defined below:

- **Verification** is intended to check that the product, service or system meets a set of design specifications. In other words, it tries to answer the question: "Are we building it right?"
- **Validation** is intended to ensure a product, service, or system meets the operational needs of the user. It addresses the question: "Are we building the right thing?” The validation process confirms that the *user requirements* have been well understood and translated into *software or system requirements* that have been implemented accordingly.

The acceptance process requires the software to be both verified and validated. Although the verification process can be performed by the development teams, validation usually requires the participation of a user representative with domain or scientific expertise.

4 Agile Projects

A development group could decide to apply an Agile development model to a request. This decision should be discussed and approved beforehand with the relevant stakeholders and DMSD management. There are several variations of Agile methods, but most Agile process models emphasize the following aspects:

1. Continuously involving a representative customer or user.
2. Developing test cases before implementing the next version of the product.
3. Implementing and testing the resulting version.
4. Demonstrating each version of the evolving product to the customer.
5. Eliciting the next requirement(s) from the customer.
6. Periodic delivery into the operational environment.

Agile development seems to be best suited to small applications projects that are conducted in the presence of a knowledgeable customer/user who has a clear understanding of the needs to be satisfied by the system that is being built. Close interaction with this user, and a high frequency (weekly or even daily) in the implementation and demonstration of the incremental modifications replace up-front requirement elicitation. There is usually no explicit design step and no design documentation. This is compensated for by a design "metaphor" that is shared among developers. A design metaphor is usually based on an architectural style, or an already existing architecture.

Note that the application of Agile methods has been limited to individual requests, not all the development activities carried on by a group. Executing a request in an Agile manner will probably be more suitable for Epics and Research Projects, because of their relative length and need for frequent interaction with a user representative.

The decision to apply Agile methods in a project should be taken carefully, judging its appropriateness with respect to the particular conditions of the project. It should be made explicit to management and stakeholders, and as said before, an available, committed and knowledgeable user representative should be assigned to the project. Time-boxing (see Section 7) is especially important for Agile projects, as in general the scope of the task won't be well defined at its initiation. It is important that an agreement of its duration and allocated resources is made up-front, and the effort shouldn't extend beyond this period. Continuing work on this project requires the submission of a new request.

5 Release Management

As far as possible, DMSD development groups will aim to establish a continuous delivery system as part of their processes. In a continuous delivery system, every change in the software triggers the execution of a continuous delivery pipeline, which as part of its initial steps compiles, builds and deploys the software. This new deployment is tested automatically. If problems are found, the developers responsible for the relevant changes are notified. If no problems are found, a new version of the software is ready to be validated (or released, if further validation is not necessary).

The goals of such a system are threefold. First, it makes every part of the process of building, deploying, testing, and releasing software visible to everybody, aiding collaboration. Second, it improves feedback so that problems are identified, and resolved, as early in the process as possible. Finally, it enables teams to deploy and release any version of their software to any supported environment through a fully automated process [3].

In some cases, development groups could decide to expose new product releases to some or all users as soon as they have been generated by the continuous delivery pipeline. In other cases where this model is impractical (like when the software needs further integration or validation tests before being deployed in production servers) only specific releases are made available.

Each development group will maintain a development plan defining the release delivery schedule for each development cycle. Each release integrates several features, and undergoes an acceptance process where additional tests are performed before the release is formally accepted. See Sections 15.7 and 15.8 for templates defining the acceptance plan and acceptance report, respectively. The Acceptance Body (see Section 8.11) is responsible for organizing the acceptance process and writing these documents. Note that a release won't be deployed permanently in production servers unless it has been accepted.

6 Software Configuration Management

Software Configuration Management is an umbrella activity that aims to identify, control, audit, and report modifications that inevitably occur on software products such as programs and associated engineering artifacts (user documentation, software requirement specifications, architecture definition documents, etc.) while they are being developed and after they have been released to customers. This activity includes keeping products under version control, defining a process to manage change requests, and reporting changes to everybody that should be informed.

This document doesn't define in full detail these processes for all DMSD groups—each group, along with its stakeholders and DMSD management, should define their required SCM activities, according to the nature of the software that is being developed, its management structures, delivery strategies and release plans, communication paths with stakeholders, etc. However, the following commonalities should be noted:

- DMSD groups have agreed to adopt Git as version control system. This system is perceived to provide better support for distributed teams, and facilitates activities associated with parallel development. The specifics about how Git will be used (what branches will be used for development, test, and production, etc.) should be defined by each group.
- Save for exceptional cases (e.g., changes in hardware or firmware or Operating System), it should always be possible to revert back to a previous release in operational environments. Releases should be identified by a unique tag.
- How formal the change control process needs to be depends on each group, but it should be accommodated as part of the workflows defined in this document. For example, a group could decide to form a Change Control Board, whose members would be responsible for approving change requests in the transition between Open and Unscheduled (where the acceptance of a ticket occurs).
- Internal reporting and coordination activities are left for each group. Cross-group coordination and external reporting are already defined (DMSD groups hold a bi-weekly coordination meeting, and submit monthly status reports to the department management).

7 Request Types and Scheduling[1]

This section describes different types of requests, and provides guidelines for how they should be *scheduled*, i.e., how each request should be assigned resources and a period of time to be executed.

By default, requests execution is *time-boxed*. This is a management technique that prioritizes schedule over deliverables (for details see [1], Appendix E). During the execution of a task, if it is anticipated that all requested deliverables will not be ready by the defined completion date, the scope of the work is reduced so that a smaller, yet still useful output is produced by such date. In other words, a task is not allowed to extend beyond its estimated completion date, preventing that delays on this task propagates to other tasks.

Although requests will be by default time-boxed, the group Lead and DMSD management can decide to adopt a different strategy, in specific cases where time-boxing is perceived as inappropriate. These decisions should be documented in the Group Management Document (in S

In general, the scheduling policy that is proposed is As Soon As Possible (ASAP), selected to favor predictability. All work is scheduled to be executed as soon as all resources necessary for its execution are available, in a first-come-first-served basis. Within this general policy, there are two exceptions:

- Blocker bugs are investigated as quickly as possible. They can interrupt other previously scheduled on-going tasks.
- Epics (just a name for relatively big projects, see Section 7.4) are scheduled in advance, and "block" resources for their estimated duration.

Each development group maintains a development "Master Schedule", a chart that makes the scheduled tasks visible to all stakeholders. In order to develop the schedule, the effort required by each request is estimated. It is suggested to perform the initial estimation by means of the "T-shirt" sizing technique (requests are classified as extra-small (XS), small (S), medium (M), large (L), or extra-large (XL), see [1], page 18).

Each development group will alternate periods of ASAP development scheduling with periods of "housekeeping", to reduce the technical debt. These housekeeping periods should be clearly indicated in the Master Schedule.

The adoption of the ASAP scheduling policy comes mainly from the CASA review. However, it is recognized that other groups may find the need to implement a different policy. For example, a group may find that a priority-based scheduling policy is more adequate for the type of requests it receives and the way they should be processed. In cases where the group Lead and DMSD management decide to apply a different scheduling policy, the selected policy should be made explicit to all stakeholders (it is included in the Group Management Document, see Section 18). Nevertheless, it is expected that each group maintains a Master Schedule.

Note: In several places in this document, a request is referred to as a *ticket*. In practical terms, requests will be managed by a commercial issue and project tracking system, JIRA (see Section 16). In this system, a request is represented by an electronic ticket. A ticket can be considered as the

1 The names for the types of requests and processing units have been modified with respect to the ones suggested in [1], to avoid name collisions and re-definitions in JIRA. No distinction has been made between request types and work types, to avoid the introduction of additional names with slightly different semantic, which could result confusing for users.

specific implementation of an abstract request, which could be implemented by other means if a different system were to be adopted (email, paper forms, etc.). As this document prescribes an implementation system for the development processes, the word "ticket" is used when discussing implementation details about managing requests.

7.1 Bug

7.1.1 Description

This is a request to correct a deviation of the system from its specified behavior. Work on bug requests follows a sequential process:

1. replicate the problem,
2. understand the problem,
3. localize the code to be repaired, and
4. depending on the problem either:
    a. repair the code or
    b. produce a workaround and create a new feature or engineering task issue to properly address the problem later, or
    c. determine for whatever reason, a fix of the bug is not needed.

7.1.2 Scheduling

If the Bug is classified as Blocker, its execution is expedited, and may interrupt other previously scheduled work if necessary. If not, it follows the normal ASAP scheduling, i.e., it may need to wait until the necessary resources become available after completing previously scheduled tasks.

7.2 Feature

7.2.1 Description

A request to change the current specified behavior. A feature could involve adding functionality to the system, or not. Although in general a Feature adds functionality, it could also involve implementing non-functional requirements, like scalability, performance, or maintainability. By definition, these do not add additional functionality to the system, but improves its properties.

7.2.2 Scheduling

Features follow ASAP scheduling. By definition, Features should have low coordination needs and low risk exposure. They represent relatively "compact" and small changes, or otherwise they should be Epics. The decision of when a request should be classified as a Feature or an Epic is left to the development team, and it may need to be modified as development moves forward. If a Feature becomes more complicated than initially thought, it may be transformed into an Epic; conversely, an Epic could become a Feature, if it ends up being simpler than originally estimated. As a guideline, the development of a Feature shouldn't take more than 1 month[2].

7.3 Engineering Task

7.3.1 Description

An internal change request to improve maintainability of the code, perform refactoring activities, clean up, improve documentation, etc. In general, Engineering Tasks won't have visible outcomes for end users, but are necessary in order to improve the quality of the system and reduce the accumulated "technical debt". It is expected that this type of request will be mostly submitted by the group development team and Architect.

7.3.2 Scheduling

In general, Engineering Tasks will be scheduled to be performed during the group housekeeping periods.

7.4 Epic[3]

7.4.1 Description

Larger cross functional work requests that require substantially more coordination than other jobs. Usually, an Epic ticket will be composed by several sub-tickets, assigned to different developers.

Epics also provide a way to manage cross-group activities. In this case, sub-tickets inside the Epic will be assigned to different groups. In the case of cross-group activities with a large scope (e.g., VLASS), it may be needed to create a special JIRA project, with a designated Lead which will be responsible for managing and tracking its activities. For all purposes, this project will be managed as a temporary development group inside DMSD.

7.4.2 Scheduling

Because Epics employ resources with different availabilities, these types of requests needs to plan ahead, when and for how long, a resource will be needed so they can make themselves available. Projects will also tend to have a larger exposure to technological and schedule risks which could have an impact on, otherwise unrelated, projects and jobs through resource dependencies.

The larger efforts that characterize a project imply that the resources working on it will not be available for other tasks for long periods of time and so the decision to proceed must be made at higher levels than in the case of other jobs. Epics will be scheduled by the PM in consultation with the Group Lead and any developers involved. Work on an Epic’s sub-tasks should not begin until the Epic itself is scheduled

Epic schedules will be time-boxed. If work is behind schedule, the scope of the Epic will be reduced

2 The limit for when a Feature becomes an Epic originated quite a bit of discussion between development groups. CASA considers the limit to be in 2 weeks. ALMA Control would prefer a longer value of 2 months. The 1-month guideline has been set as the average between these two values, rounded to entire months.

3 The word "Epic" comes from Scrum, and it is the name adopted by JIRA for this type of request.

in such a way that maximizes the impact of the project without causing a schedule delay.

7.5 Research request

7.5.1 Description

Research requests are submittals for new software capabilities in which either the requester cannot define in objective terms what is the expected result, or the developers are unsure about whether the requirement can be implemented within the known capabilities and limitations of the system and its operating environment.

7.5.2 Scheduling

These requests have two goals: (1) finding what is sought; and (2) learning. In order to mitigate the risk of these projects to delay other efforts and waste resources if neither of these goals can be reached, these research projects will be time-boxed. In addition, the risk of wasting resources will be mitigated by interspersing "tollgates" in the workflow.

Tollgates are pre-established decision points in the life of a project. At each tollgate, the project will be reviewed from three different perspectives: science, progress and cost. Depending on the results of this review, a decision is made on whether to continue with the project, abandon it, defer it, or submit a follow-up request.

8 Roles and Responsibilities

8.1 Reporter

This is the person who creates the ticket, and who has an interest in the activity.

Save for exceptional cases, developers shouldn't create tickets on behalf of somebody else, although they can create tickets on their own behalf. A ticket needs to be created by the interested party. This is necessary so the system effectively maps the Reporter role with the particular person that assumed this role in the context of a ticket, facilitating the execution of processes associated with the request (e.g., correctly sending questions and notifications to the actual Reporter, and not to the person who created the ticket).

8.2 Project Scientist

The project scientist is the main interface between the scientific stakeholders and the development group. His main responsibilities includes: to judge the worthiness of change and research proposals, to work with users to clarify requests and with developers to explain the goals. The project scientist must ensure that the criteria for accepting features are specified and the tests that verify those criteria are later run to determine whether the features have been completed satisfactorily. A key requirement of this position is availability.

8.3 Group Lead

The Group Lead is responsible for the overall group management.

8.4 Component Lead

Some development groups (noticeable CASA) are big enough to require being subdivided in *components*. In this case, the Component Lead is assigned to support the Group Lead on the activities specific for a component.

8.5 Component Scientist

Some development groups (noticeable CASA) are big enough to require being subdivided in *components*. In this case the Component Scientist assumes the responsibly of the Project Scientist, for the activities pertaining to a component.

8.6 Project Manager

In general the Project Manager is responsible for all the activities related with managing projects, applying processes, methods, knowledge, skills and experience to achieve the specified objectives. His specific responsibilities should be discussed and agreed with the Group Lead, and generally will include to make the initial assignment of requests, enforce the workload management policy, consolidate all the scheduling, resource calendar and progress information, and serve as center of excellence in project management and system engineering, assisting other members of the group with requirement elicitation, estimation, risk management, planning, quality assurance, verification testing, validation testing, and process definition as needed.

8.7 Software Architect

The main responsibility of the Software Architect is to define, document and oversee the implementation of the software architecture.

The Architect also performs studies to assess the feasibility of possible technologies and evaluates their relative advantages and drawbacks. Working with the Test Group, it performs scalability and performance tests to validate architectural choices.

There are two architecture-related roles: an intra-group Architect that takes care of the above responsibilities inside a group; and a multi-group Architect, that takes care of architectural concerns that span more than one group, and evaluates group specific architectures.

8.8 Developer

The Developer is responsible for the development of new functionality, and the maintenance of the existing code base. The Developer provides early effort estimation, analyzes requirements, designs, implements and tests changes in the system.

8.9 Test Engineer

The verification of some features may require additional tests that extend the unit tests implemented by developers. For instance, this is usually required for systems that integrate several components, developed by different developers and groups. The Test Engineer will analyze requirements and their implementation, and defines additional tests, addressing integration concerns and qualities such as scalability, performance, and usability. It ensures that the system being developed is testable, and advices on testing strategies and frameworks. This role can be assumed by members of DMSD Test Group, or by designated members of the development group. It can be a part-time job.

8.10 Validator

This is who validates a task and confirms that the implemented changes comply with requirements, and the requirements faithfully represent the stakeholder's needs (see section 3, "Software Testing Verification and Validation").   This role will normally be assumed by the Reporter or the Project Scientist, but it could be assigned to somebody else if appropriate.

8.11 Acceptance Body

This is the nominated person or group who has the responsibility of accepting a *release*. This role is assigned by common agreement between DMSD management and the relevant stakeholders. A release integrates several features together, and delivers them as a package. Note that this role is different from the Validator, who validates at the task level.

Most of the work can be handled at the task level, with the aid of continuous integration strategies (see Section 5, "Release Management"). However, it could be necessary to deliver new software in releases, with formal acceptances. This is typically required when systems are deployed in production servers, in systems where downtime costs are high, and therefore the new deployment needs to be carefully tested to prevent integration issues.

A new release should not be deployed in production unless it has been accepted by the Acceptance Body. Templates for the Acceptance Plan and Acceptance Report can be found in Sections 15.7 and 15.8.

Quality Assurance Role

The responsibility of the quality assurance (QA) role is not to test the software. Its role is to educate and ensure conformance to the development practices the organization choose for itself. Issues identified by QA must first be addressed within the group but, if for whatever reason, this is not possible they should be escalated for resolution. The QA role could be performed by a dedicated resource, by rotating the role among developers or by assigning it to the Project Manager.

8.12 Expert

At several points in the workflows, it could be necessary to complete missing information, clarify details, and in general request information to the most appropriate or knowledgeable person. This is the Expert. Depending on the type of requested information, this role can be assumed by the Project Scientist, the Reporter, the Software Architect, or any other person who is capable of providing the required information. Note that this is not a static role in the workflows, but varies depending on the type of information that is being requested. For questions pertaining to the science requirements, this role is typically assumed by the Project Scientist.

9 Bug Workflow

The workflow for Bug requests is presented in Figure 1.

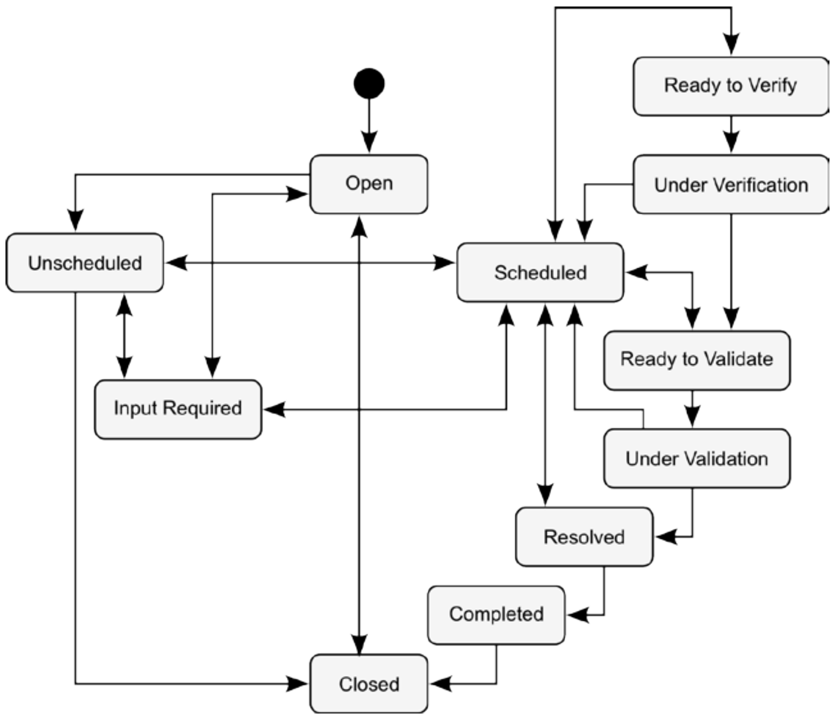


Figure 1: Bug workflow.

9.1 State: Open

9.1.1 Purpose

This is the initial state after a new ticket is created by Reporter. The main activity that needs to be performed while in this state is to review that there is enough supporting information for the bug to be processed in the next steps.

9.1.2 Responsible Role

Group Lead or Component Lead.

9.1.3 Inputs

It is expected that Reporter, who creates the ticket, has provided the following inputs:

- Bug description, including environment, release affected, etc.
- Bug priority.
- Test description, documenting how the problem can be reproduced.

9.1.4 Outputs

- Revised bug priority.
- Accepted or rejected decision, with accompanying explanatory comment.

9.1.5 Transitions

- To **Input Required**, if additional information is required. The ticket is assigned to the most appropriate person to provide an answer, the Expert.
- To **Unscheduled**, if the Bug is accepted.
- To **Closed**, if after analyzing the problem, it is decided that no work needs to be scheduled. For example, in case the issue reported is not actually a problem, or if it is duplicated.

9.1.6 Tollgates

- **Completeness of bug description and test procedure.** These artifacts need to be provided in sufficient detail for the bug to be investigated. If this information is not provided within a reasonable time (guideline: 1 month), the ticket is closed.

9.2 State: Unscheduled

9.2.1 Purpose

This state indicates that the ticket has been accepted, but hasn't been scheduled yet. In Agile methodologies, the collection of tickets in this state is referred as the **"backlog"**. The main operation to be performed in this state is for Developer to estimate the effort and schedule the task.

9.2.2 Responsible Role

Developer.

9.2.3 Inputs

None. The information necessary to proceed with the effort estimation and scheduling should have been already provided.

9.2.4 Outputs

- Effort estimation. The T-shirt technique is recommended as a way to come up with an initial rough estimation.
- Schedule, specifying a start and end date.

9.2.5 Transitions

- To **Closed**, if Developer finds out that no work is actually needed.
- To **Scheduled**, when a start and end date has been assigned for the ticket.

- To **Input Required**, if Developer finds that additional information or clarifications are needed in order to estimate the effort.

9.2.6 Tollgates

None.

9.3 State: Input Required

9.3.1 Purpose

This state is used to indicate that the ticket requires additional information.

9.3.2 Responsible Role

Expert.

9.3.3 Inputs

- A clear statement specifying what information is required, and why it's necessary to continue with the process. In addition, it is also recommended to specify a deadline for when the information should be provided.

9.3.4 Outputs

- The required information.

9.3.5 Transitions

- Back to **Open**, after providing the requested information.
- Back to **Unscheduled**, after providing the requested information.
- Back to **Scheduled**, after providing the requested information.

9.3.6 Tollgates

- The requested information needs to be provided in a reasonable time (guideline: 1 month), taking into account that most of the requests will be time-boxed. If the information cannot be provided, the ticket may need to be un-scheduled or closed.

9.4 State: Scheduled

9.4.1 Purpose

This is the state where the bug is investigated, reproduced, fixed and tested. User documentation should be updated, when needed.

9.4.2 Responsible Role

Developer.

9.4.3 Inputs

- Bug documentation, which should have been already analyzed and clarified while in the Open state.
- Test description, specifying how the bug can be reproduced.

9.4.4 Outputs

- Bug fixes.
- A note documenting what was the problem, and summarizing the implemented solution.
- If the solution provides a workaround, but doesn't address the root cause of the problem, it may be necessary to submit additional tickets. These should be referenced in the current ticket.

9.4.5 Transitions

- To **Input Required**, if during the implementation the developer finds out that additional information is required. The ticket is assigned to Expert.
- To **Ready to Verify**, if the bug requires to be verified in integration. In this case, the feature is assigned to Test Engineer.
- To **Ready to Validate**, once the bug has been fixed and tested, and has been merged into the proper branch. The ticket is assigned to Validator.
- To **Unscheduled**, if the investigation needs to be stopped (because of the submission of a Blocker bug, for example), or if the time box allocated for this task has expired. This transition is also necessary to deal with process errors (i.e., the ticket was moved to Scheduled by mistake).
- To **Resolved**, if the bug is simple enough (correct a typo in a GUI label, for example) that doesn't require the participation of a specialized Validator. This is a concession for the fact that development and science groups have scarce resources, which shouldn't be wasted unnecessarily.

9.4.6 Tollgates

None

9.5 State: Ready to Verify

9.5.1 Purpose

This is an **optional** state, for bugs that require additional verification tests besides the ones executed by Developer while in the Scheduled state. During the Ready to Verify state, the problem is analyzed and additional tests are specified.

This may be necessary for several reasons:

- The verification of some bugs may require to be executed against telescope hardware, while Developer may use hardware simulators instead. In this case, the fix will need to be tested against real hardware to complete the verification.
- Some bugs may require to be executed in an integration setup, while Developer may use mockups to replace components that are outside his area of responsibility. One example of this is the use of databases, or other shared distributed services that are impractical to be maintained by each developer.
- The fix for a bug may break other functionality. An automatic test suite running in a continuous integration server may detect these problems, if the system has sufficient test coverage. If not, additional tests may be necessary.

This separate verification step allows for specialized Test Engineers (which can be part of the development group, or a separate, DMSD-wide group) to analyze requirements and ensure that the feature has been sufficiently tested, addressing concerns that may not have been taken into consideration during development, such as scalability, usability, etc.

9.5.2 Responsible Role

Test Engineer.

9.5.3 Inputs

- Bug description.
- Bug test description, documenting how the problem can be reproduced.

9.5.4 Outputs

- Additional test specifications.

9.5.5 Transitions

- To **Under Verification**, once the resources necessary to perform the additional test activities are available.

9.5.6 Tollgates

None

9.6 State: Under Verification

9.6.1 Purpose

Execute and report the results of additional verification tests.

9.6.2 Responsible Role

Test Engineer.

9.6.3 Inputs

- Additional test specifications.

9.6.4 Outputs

- Test reports.

9.6.5 Transitions

- To **Scheduled**, if a problem has been found in the tests, which require the attention of Developer. The ticket is assigned to Developer.
- To **Ready to Validate**, if tests have passed satisfactorily, and the bug is ready to be validated.

9.6.6 Tollgates

None

9.7 State: Ready to Validate

9.7.1 Purpose

This is a waiting state, where the bug is ready to be tested by Validator (usually the Reporter himself).

9.7.2 Responsible Role

Validator.

9.7.3 Inputs

- Test procedure, provided when the bug ticket was created.

9.7.4 Outputs

- Validation Report.

9.7.5 Transitions

- To **Under Validation**, when Validator is ready to start the validation tests.
- Back to **Scheduled**, if Validator decides that the ticket is not ready yet for validation. In this case, he should annotate explicitly what is missing. This transition is also provided to account for process errors.

9.7.6 Tollgates

None.

9.8 State: Under Validation

9.8.1 Purpose

The purpose of this state is to perform the validation tests, validating that the bug has been effectively fixed.

9.8.2 Responsible Role

Validator.

9.8.3 Inputs

- Test procedure, provided when the bug ticket was created.

9.8.4 Outputs

- Test Report, documenting the results of the tests executed.

9.8.5 Transitions

- To **Resolved**, if the validation tests demonstrate that the bug has been fixed. The ticket is assigned to Developer.
- Back to **Scheduled**, if tests show that the problem hasn't been fixed, and the ticket requires additional work by Developer. The ticket is assigned to Developer.

9.8.6 Tollgates

None

9.9 State: Resolved

9.9.1 Purpose

In the Resolved state, the Developer performs post-development activities, such as merging to the final branch, etc. This is also the state where reviews and quality control can be introduced.

9.9.2 Responsible Role

Developer.

9.9.3 Inputs

None.

9.9.4 Outputs

None.

9.9.5 Transitions

- To **Completed**, once post-development activities have been completed.
- To **Scheduled**, to allow for errors, i.e., the ticket was moved to Resolved before it was ready.

9.9.6 Tollgates

None.

9.10 State: Completed

9.10.1 Purpose

The ticket is in the Complete state until it has been delivered. In general, all remaining activities necessary to finalize the activity should be performed while in this state. This could involve the creation of a release branch, the deployment in production, etc.

9.10.2 Responsible Role

Developer.

9.10.3 Inputs

None.

9.10.4 Outputs

- Associated artifacts necessary to finalize the activity, such as documentation updates, configuration artifacts, etc.

9.10.5 Transitions

- To **Closed**, once the bug has been delivered.

9.11 **State: Closed**

9.11.1 Purpose

This is the final state. No more work can be done in the context of this ticket. If additional work is required, a new ticket should be created.

9.11.2 Transitions

- To **Open**, to allow for errors, i.e., the ticket was closed before it was ready. This transition may be used by project administrators only.

10 Feature Workflow

The workflow for Features is presented in Figure 2. The Feature Workflow is very similar to the Bug Workflow. The main difference is in the artifacts required by each step.

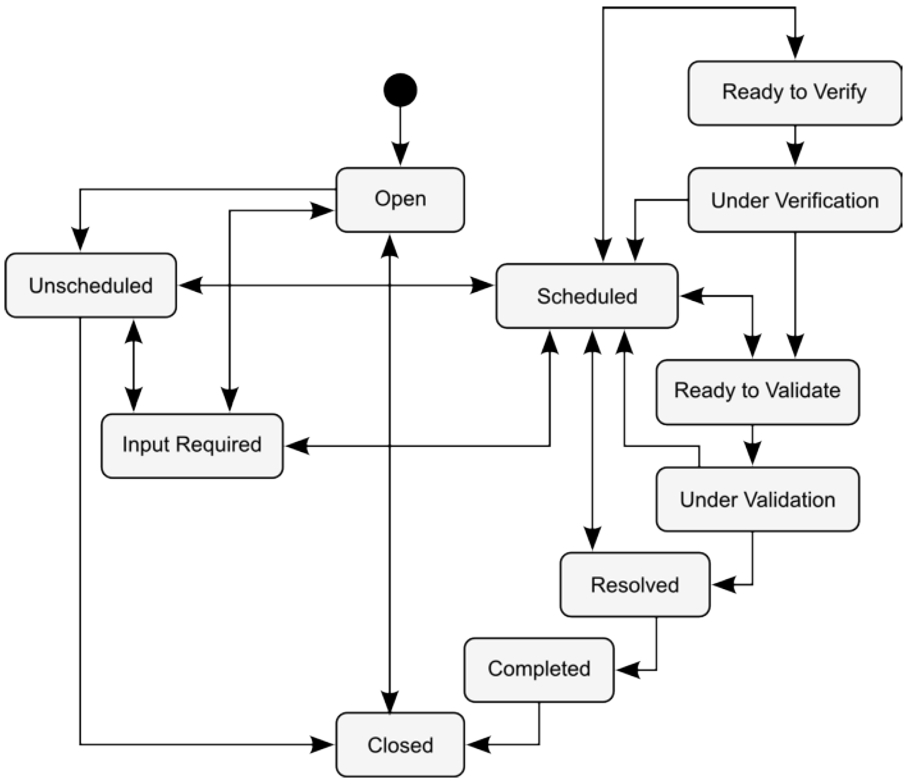


Figure 2: Feature workflow.

10.1 State: Open

10.1.1 Purpose

This is the initial state after a new ticket is created by Reporter. The main activity that needs to be performed while in this state is to review that the supporting information is complete enough to proceed with the next steps.

10.1.2 Responsible Role

Group Lead or Component Lead.

10.1.3 Inputs

It is expected that Reporter, who creates the ticket, has provided the following artifacts:

- Software Requirements Specification.
- Validation Plan.

10.1.4 Outputs

- Accepted or rejected decision, with accompanying explanatory comment.

10.1.5 Transitions

- To **Input Required**, if additional information is required. The ticket is assigned to the most appropriate person to provide an answer, the Expert.
- To **Unscheduled**, if the Feature is accepted.
- To **Closed**, if after reviewing the requested Feature, it is decided that the ticket should be rejected. This transition is accompanied by an explanatory comment. A Closed ticket can be re-opened if, upon a request from Reporter, it is decided to revisit this decision.

10.1.6 Tollgates

- **Completeness of Software Requirements Specification and Validation Plan.** These artifacts need to be provided in sufficient detail for the Feature to be implemented. If this information is not provided within a reasonable time, the ticket is closed.

Note that a change request for which it is not possible to define a Software Requirements Specification and Validation Plan cannot be submitted as a Feature. It can be submitted as a Research Project, on the other hand.

10.2 State: Unscheduled

10.2.1 Purpose

This state indicates that the ticket has been accepted, but hasn't been scheduled yet. In Agile methodologies, the collection of tickets in this state is referred as the **"backlog"**. The main operation necessary in this state is for Developer to perform the effort estimation and schedule the task.

10.2.2 Responsible Role

Developer.

10.2.3 Inputs

- Software Requirements Specification, provided when the ticket was created.
- Validation Plan, provided when the ticket was created.

10.2.4 Outputs

- Effort estimation. The T-shirt technique can be used for the purpose of an early, rough estimation.

- Schedule, specifying a start and end date.

10.2.5 Transitions

- To **Closed**, if Developer finds out that no work is actually needed. For example, he/she could find that the request is not feasible, or that the ticket is duplicated.
- To **Scheduled**, when a start and end date has been assigned for the ticket.
- To **Input Required**, if Developer finds that additional information or clarifications are needed in order to estimate the effort.

10.2.6 Tollgates

None

10.3 State: Input Required

10.3.1 Purpose

This state is used to indicate that the ticket requires additional information.

10.3.2 Responsible Role

Expert.

10.3.3 Inputs

- A clear statement specifying what information is required, and why it is necessary to continue with the process.

10.3.4 Outputs

- The missing information.

10.3.5 Transitions

- Back to **Open**, after providing the requested information.
- Back to **Unscheduled**, after providing the requested information.
- Back to **Scheduled**, after providing the requested information.

10.3.6 Tollgates

- The requested information needs to be provided in reasonable time, taking into account that most of the requests will be time-boxed. If the information cannot be provided, the ticket may need to be un-scheduled or closed.

10.4 State: Scheduled

10.4.1 Purpose

This is the state where the Feature is implemented, documented, and tested.

10.4.2 Responsible Role

Developer.

10.4.3 Inputs

- Software Requirements Specification, provided when the ticket was created.
- Validation Plan, provided when the ticket was created.

10.4.4 Outputs

- Feature Implementation.
- Unit Tests.
- Additional implementation details (documented as comments).
- User documentation

10.4.5 Transitions

- To **Input Required**, if during the implementation the developer finds out that additional information is required. The ticket is assigned to Expert.
- To **Ready to Verify**, if the Feature requires to be verified in integration. In this case, the feature is assigned to Test Engineer.
- To **Ready to Validate**, when the Feature has been implemented and tested, and has been merged into the proper branch for testing. The ticket is assigned to Validator.
- To **Unscheduled**, if the investigation needs to be stopped (because of the submission of a Blocker bug, for example), or if the time box allocated for this task has expired. This transition is also necessary to deal with process errors (i.e., the ticket was moved to Scheduled by mistake).
- To **Resolved**, if the Feature is simple enough (change a GUI label, for example) that doesn't require the participation of a specialized Validator. This is a concession for the fact that development and science groups have scarce resources, which shouldn't be waisted unnecessarily.

10.4.6 Tollgates

Note

10.5 State: Ready to Verify

10.5.1 Purpose

This is an optional state, for Features that require additional verification tests besides the ones executed by Developer while in the Scheduled state. During the Ready to Verify state, the problem is analyzed by Test Engineer and additional tests are specified.

See Section 9.5 for additional details about this state.

10.5.2 Responsible Role

Test Engineer.

10.5.3 Inputs

- Software Requirements Specification, provided when the ticket was created.
- Validation Plan, provided when the ticket was created.
- Additional implementation information (the code itself, unit tests, implementation comments).

10.5.4 Outputs

- Additional Test Plans.

10.5.5 Transitions

- To **Under Verification**, once the additional Test Plans have been specified, and the resources necessary to perform the additional test activities are available.

10.5.6 Tollgates

None.

10.6 State: Under Verification

10.6.1 Purpose

Execute verification tests and report their results.

10.6.2 Responsible Role

Test Engineer.

10.6.3 Inputs

- Additional test specifications.

10.6.4 Outputs

- Test Reports (see Section 15.4)

10.6.5 Transitions

- To **Scheduled**, if a problem has been found in the verification tests, which require the attention of Developer. The ticket is assigned to Developer.
- To **Ready to Validate**, if tests have passed satisfactorily and the bug is ready to be validated.

10.6.6 Tollgates

None

10.7 State: Ready to Validate

10.7.1 Purpose

This is a waiting state, where the feature is ready to be tested by Validator (usually the Reporter himself).

10.7.2 Responsible Role

Validator.

10.7.3 Inputs

- Validation Plan, provided when the bug ticket was created.

10.7.4 Outputs

- None.

10.7.5 Transitions

- To **Under Validation**, when Validator is ready to start the validation tests.
- Back to **Scheduled**, if Validator decides that the ticket is not ready yet for validation. In this case, he should annotate explicitly what is missing. This transition is also provided to account for process errors.

10.7.6 Tollgates

None.

10.8 State: Under Validation

10.8.1 Purpose

The purpose of this state is to perform the validation tests, checking that the Feature complies with the specified requirements.

10.8.2 Responsible Role

Validator.

10.8.3 Inputs

- Validation Plan, provided when the bug ticket was created.

10.8.4 Outputs

- Validation Report, documenting the results of the validation tests that were executed.

10.8.5 Transitions

- To **Resolved**, if the validation tests demonstrate that Feature complies with the specified requirements. The ticket is assigned to Developer.
- Back to **Scheduled**, if tests show that the Feature as implemented doesn't comply with the specified requirements, and the ticket requires additional work. The problem needs to be clearly specified with a comment and the ticket assigned back to Developer. Developer needs to evaluate if the reported problem exposes an implementation problem —in which case it should just be corrected—, or imposes a significant change or extension of the original requirements. In this case, the original effort estimation is invalid, so the ticket should be moved to Unscheduled, and the workflow is interrupted.

10.8.6 Tollgates

None.

10.9 State: Resolved

10.9.1 Purpose

In the Resolved state, the Developer performs post-development activities, such as merging to the final branch, etc. This is also the state where reviews and quality control can be introduced.

10.9.2 Responsible Role

Developer.

10.9.3 Inputs

None.

10.9.4 Outputs

None.

10.9.5 Transitions

- To **Completed**, once post-development activities have been completed.
- To **Scheduled**, to allow for errors, i.e., the ticket was moved to Resolved before it was ready.

10.9.6 Tollgates

None.

10.10 State: Completed

10.10.1 Purpose

The ticket is in the Complete state until it has been delivered. In general, all remaining activities necessary to finalize the activity should be performed while in this state. This could involve the creation of a release branch, the deployment in production, documentation updates, etc.

10.10.2 Responsible Role

Developer.

10.10.3 Inputs

None.

10.10.4 Outputs

- Associated artifacts necessary to finalize the activity, such as documentation updates, configuration artifacts, etc.

10.10.5 Transitions

- To **Closed**, once the Feature has been delivered.

10.10.6 Tollgates

None.

10.11State: Closed

10.11.1Purpose

This is the final state. No more work can be done in the context of this ticket. If additional work is required, a new ticket should be created.

10.11.2Transitions

- To **Open**, to allow for errors, i.e., the ticket was closed before it was ready. This transition may be used by project administrators only.

11 Engineering Task Workflow

The Engineering Task workflow is presented in Figure 3. The main difference with the Bug and Feature workflows is the absence of validation steps.

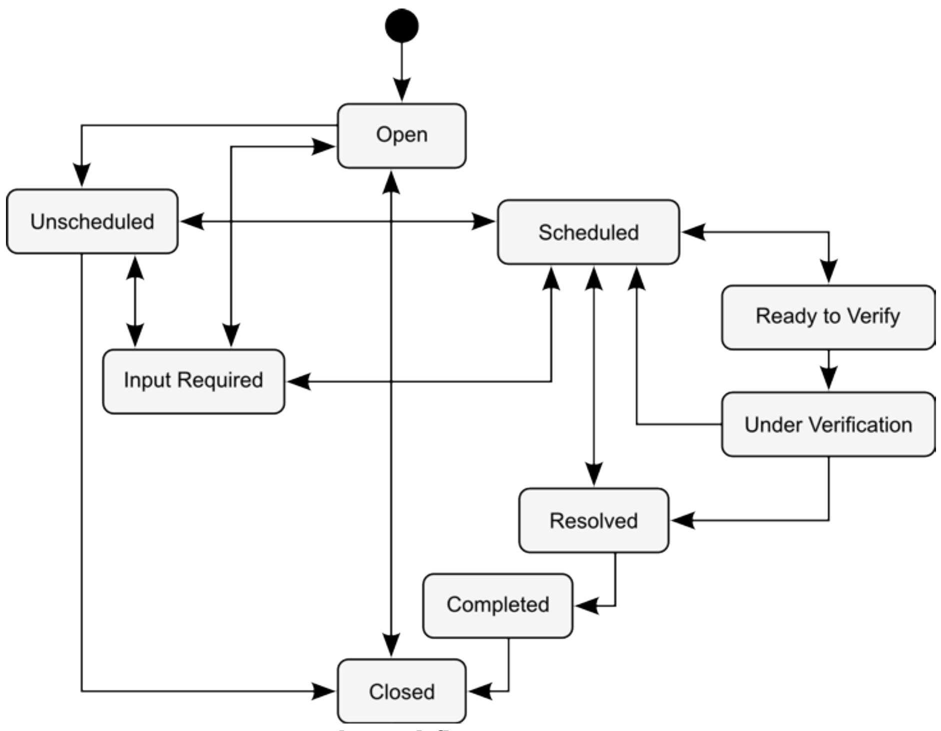


Figure 3: Engineering Task Workflow

11.1 State: Open

11.1.1 Purpose

This is the initial state after a new ticket is created by Reporter. The main activity that needs to be performed while in this state is to review that there is enough supporting information for the request to proceed with the next steps.

11.1.2 Responsible Role

Group Lead or Component Lead.

11.1.3 Inputs

It is expected that Reporter, who creates the ticket, has provided the following inputs:

- Software Requirement Specification.
- Test Plan.

11.1.4 Outputs

- Accepted or rejected decision, with accompanying explanatory comment.

11.1.5 Transitions

- To **Input Required**, if additional information is required. The ticket is assigned to the most appropriate person to provide an answer, the Expert.
- To **Unscheduled**, if the Engineering Task is accepted.
- To **Closed**, if after analyzing the request, it is decided that no work needs to be scheduled. For example, the request could be duplicated.

11.1.6 Tollgates

None.

11.2 State: Unscheduled

11.2.1 Purpose

This state indicates that the ticket has been accepted, but hasn't been scheduled yet. In Agile methodologies, the collection of tickets in this state is referred as the **"backlog"**. The main operation necessary in this state is for Developer to perform the effort estimation and schedule the task.

11.2.2 Responsible Role

Developer.

11.2.3 Inputs

None. The information necessary to proceed with the effort estimation and scheduling should have been already provided.

11.2.4 Outputs

- Effort estimation. The T-shirt technique can be used for the purpose of an early estimation.

- Schedule, specifying a start and end date.

11.2.5 Transitions

- To **Closed**, if Developer finds out that no work is actually needed.
- To **Scheduled**, when a start and end date has been assigned for the ticket.
- To **Input Required**, if Developer finds that additional information or clarifications are needed in order to estimate the effort.

11.2.6 Tollgates

None.

11.3 State: Input Required

11.3.1 Purpose

This state is used to indicate that the ticket requires additional information.

11.3.2 Responsible Role

Expert.

11.3.3 Inputs

- A clear statement specifying what information is required, and why it's necessary to continue with the process.

11.3.4 Outputs

- The missing information.

11.3.5 Transitions

- Back to **Open**, after providing the requested information.
- Back to **Unscheduled**, after providing the requested information.
- Back to **Scheduled**, after providing the requested information.

11.3.6 Tollgates

None.

11.4 State: Scheduled

11.4.1 Purpose

This is the state where the Engineering Task is performed.

11.4.2 Responsible Role

Developer.

11.4.3 Inputs

- Software Requirements Specification.
- Test Plan.

11.4.4 Outputs

- Implementation, which depends on the type of task.

11.4.5 Transitions

- To **Input Required**, if during the implementation the developer finds out that additional information is required. The ticket is assigned to Expert.
- To **Ready to Verify**, if the Engineering Task requires to be verified in integration. In this case, the feature is assigned to Test Engineer.
- To **Unscheduled**, if the task needs to be stopped (because of the submission of a Blocker bug, for example), or if the time box allocated for this task has expired. This transition is also necessary to deal with process errors (i.e., the ticket was moved to Scheduled by mistake).
- To **Resolved**, if the Engineering Task doesn't require additional integration verification tests, besides the unit tests performed by Developer.

11.4.6 Tollgates

None.

11.5 State: Ready to Verify

11.5.1 Purpose

This is an optional state, for bugs that require additional verification tests besides the ones executed by Developer while in the Scheduled state. During the Ready to Verify state, the problem is analyzed and additional tests are specified.

See Section 9.5 for additional details about this state.

11.5.2 Responsible Role

Test Engineer.

11.5.3 Inputs

- Software Requirements Specification.
- Test Plan.

11.5.4 Outputs

- Additional test specifications.

11.5.5 Transitions

- To **Under Verification**, once the resources necessary to perform the additional test activities are available.

11.5.6 Tollgates

None.

11.6 State: Under Verification

11.6.1 Purpose

Execute and report the results of additional verification tests.

11.6.2 Responsible Role

Test Engineer.

11.6.3 Inputs

- Additional test specifications.

11.6.4 Outputs

- Test reports.

11.6.5 Transitions

- To **Scheduled**, if a problem has been found in the tests, which require the attention of Developer. The ticket is assigned to Developer.
- To **Resolved**, if test results are satisfactory.

11.6.6 Tollgates

None.

11.7 State: Resolved

11.7.1 Purpose

In the Resolved state, the Developer performs post-development activities, such as merging to the final branch, etc. This is also the state where reviews and quality control can be introduced.

11.7.2 Responsible Role

Developer.

11.7.3 Inputs

None.

11.7.4 Outputs

None.

11.7.5 Transitions

- To **Completed**, once post-development activities have been completed.
- To **Scheduled**, to allow for errors, i.e., the ticket was moved to Resolved before it was ready.

11.7.6 Tollgates

None.

11.8 State: Completed

11.8.1 Purpose

The ticket is in the Complete state until it has been delivered. In general, all remaining activities necessary to finalize the activity should be performed while in this state. This could involve the creation of a release branch, the deployment in production, documentation updates, etc.

11.8.2 Responsible Role

Developer.

11.8.3 Inputs

None.

11.8.4 Outputs

- Associated artifacts necessary to finalize the activity, such as documentation updates, configuration artifacts, etc.

11.8.5 Transitions

- To **Closed**, once the bug has been delivered.

11.8.6 Tollgates

None.

11.9 **State: Closed**

11.9.1 Purpose

This is the final state. No more work can be done in the context of this ticket. If additional work is required, a new ticket should be created.

11.9.2 Transitions

- To **Open**, to allow for errors, i.e., the ticket was closed before it was ready. This transition may be used by project administrators only.

12 **Epic Workflow**

The Epic Workflow is presented in Figure 4. Epics represent significant development activities, typically broken down in sub-features. An Epic parent ticket represents the whole effort, while child tickets are created for each sub-feature. The workflow described in this section relate to the Epic, while sub-features follow the Feature Workflow.

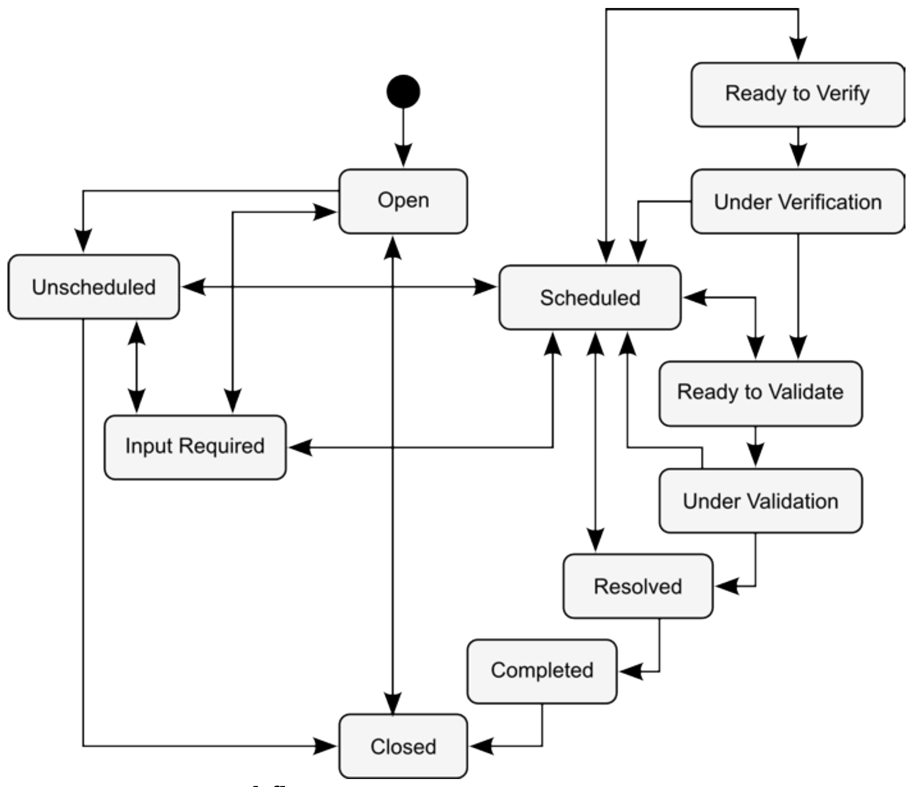


Figure 4: Epic Workflow.

12.1 State: Open

12.1.1 Purpose

This is the initial state after a new ticket is created by Reporter. The main activity that needs to be performed while in this state is to review that there is enough supporting information for the Epic to be processed in the next steps.

12.1.2 Responsible Role

Group Lead or Component Lead.

12.1.3 Inputs

It is expected that Reporter, who creates the ticket, has provided the following inputs:

- Software Requirements Specification, for each sub-feature.
- Validation Plan, for each sub-feature.

12.1.4 Outputs

- Accepted or rejected decision, with accompanying explanatory comment.

12.1.5 Transitions

- To **Input Required**, if additional information is required. The ticket is assigned to the most appropriate person to provide an answer, the Expert.
- To **Unscheduled**, if the Epic is accepted.
- To **Closed**, if it has been decided to reject the Epic.

12.1.6 Tollgates

- **Completeness of each Sub-feature Software Requirements Specification and Validation Plan.** These artifacts need to be provided in sufficient detail for the Epic to be performed. If this information is not provided within a reasonable time (1 month can be used as guideline), the ticket is closed.

12.2 State: Unscheduled

12.2.1 Purpose

This state indicates that the ticket has been accepted, but hasn't been scheduled yet. In order for the Epic to be scheduled, the effort estimation for each sub-features should be performed. Once this is done, the Group Lead reviews the sub-feature estimations and schedules the Epic.

12.2.2 Responsible Role

Project Manager.

12.2.3 Inputs

- Effort estimations for each one of the sub-features in the Epic.
- Sub-feature dependency analysis.

12.2.4 Outputs

- Project Plan.
- Epic Schedule, specifying a start and end date.
- Sub-feature assignment and schedules.

12.2.5 Transitions

- To **Closed**, if it is decided that the Epic will not be executed.
- To **Scheduled**, when a start and end date has been assigned to the Epic.
- To **Input Required**, if Group Lead finds that additional information or clarifications is needed in order to schedule the Epic.

12.2.6 Tollgates

None.

12.3 State: Input Required

12.3.1 Purpose

This state is used to indicate that the ticket requires additional information.

12.3.2 Responsible Role

Expert.

12.3.3 Inputs

- A clear statement specifying what information is required, and why it's necessary to continue with the process.

12.3.4 Outputs

- The missing information.

12.3.5 Transitions

- Back to **Open**, after providing the requested information.
- Back to **Unscheduled**, after providing the requested information.
- Back to **Scheduled**, after providing the requested information.

12.3.6 Tollgates

None.

12.4 State: Scheduled

12.4.1 Purpose

This is the state where the Epic work is carried on. Development is performed in each one of the sub-features. The main activity for the Epic, on the other hand, is monitoring and reporting on the whole effort, as necessary.

12.4.2 Responsible Role

Project Manager.

12.4.3 Inputs

- Progress reports for each sub-feature.

12.4.4 Outputs

- Epic progress report.

The format of this artifact is left at the discretion of the Group Lead. The Group Lead should agree on the format and frequency of reporting with upper management and other stakeholders. One possible (brief) format for this type of report, used so far in DMSD, are itemized lists for "Progress", "Next Tasks", and "Issues".

12.4.5 Transitions

- To **Input Required**, if during the implementation the developer finds out that additional information is required. The ticket is assigned to Expert.
- To **Ready to Verify**, if the Epic requires to be verified in integration. In this case, the Epic is assigned to Test Engineer.
- To **Ready to Validate**, once all sub-features in the Epic has been implemented and tested, and have been merged into a Test branch. The ticket is assigned to Validator.
- To **Unscheduled**, if the Epic needs to be stopped (because of the submission of a Blocker bug, for example), or if the time box allocated for this task has expired. This transition is also necessary to deal with process errors (i.e., the ticket was moved to Scheduled by mistake).
- To **Resolved**, if the Epic sub-tasks are simple enough that don't require the participation of a specialized Validator.

12.4.6 Tollgates

None.

12.5 State: Ready to Verify

12.5.1 Purpose

This is an optional state, for Epics that require additional verification tests besides the ones executed by Developer while in the Scheduled state. During the Ready to Verify state, the problem is analyzed and additional tests are specified.

See Section 9.5 for additional details about this state.

12.5.2 Responsible Role

Test Engineer.

12.5.3 Inputs

- Sub-feature information.

12.5.4 Outputs

- Additional test specifications.

12.5.5 Transitions

- To **Under Verification**, once the resources necessary to perform the additional test activities are available.

12.5.6 Tollgates

None.

12.6 State: Under Verification

12.6.1 Purpose

Execute and report the results of additional verification tests.

12.6.2 Responsible Role

Test Engineer.

12.6.3 Inputs

- Additional test specifications.

12.6.4 Outputs

- Test reports.

12.6.5 Transitions

- To **Scheduled**, if a problem has been found in the tests. In this case specific sub-features are assigned to the corresponding Developer, and the Epic is assigned to the Group Lead.
- To **Ready to Validate**, if tests have passed satisfactorily, and the Epic is ready to be validated.

12.6.6 Tollgates

None.

12.7 State: Ready to Validate

12.7.1 Purpose

This is a waiting state, where the bug is ready to be tested by Validator (usually the Reporter himself).

12.7.2 Responsible Role

Validator.

12.7.3 Inputs

- Test procedure, provided when the bug ticket was created.

12.7.4 Outputs

- Validation Report.

12.7.5 Transitions

- To **Under Validation**, when Validator is ready to start the validation tests.
- Back to **Scheduled**, if Validator decides that the ticket is not ready yet for validation. In this case, he should annotate explicitly what is missing. This transition is also provided to account for process errors.

12.7.6 Tollgates

None.

12.8 State: Under Validation

12.8.1 Purpose

The purpose of this state is to perform the validation tests, validating that the Epic sub-features comply with their specified requirements.

12.8.2 Responsible Role

Validator.

12.8.3 Inputs

- Validation Plans, for each sub-feature.

12.8.4 Outputs

- Validation Reports, for each sub-feature.

12.8.5 Transitions

- To **Resolved**, if the validation tests demonstrate that the Epic sub-features comply with their specified requirements. The Epic ticket is assigned to the Project Manager, to coordinate any additional required work.
- Back to **Scheduled**, if tests demonstrate problems in the implementation. Sub-features are assigned back to the corresponding developer. The Epic ticket is assigned to the Project Manager.

12.8.6 Tollgates

None.

12.9 State: Resolved

12.9.1 Purpose

In the Resolved state, the Project Manager coordinates any required post-development work in the sub-features, like merging to the final branch, etc. This is also the state where reviews and quality control can be introduced.

12.9.2 Responsible Role

Project Manager.

12.9.3 Inputs

None.

12.9.4 Outputs

None.

12.9.5 Transitions

- To **Completed**, once post-development activities have been completed.
- To **Scheduled**, to allow for errors, i.e., the ticket was moved to Resolved before it was ready.

12.9.6 Tollgates

None.

12.10State: Completed

12.10.1Purpose

The ticket is in the Complete state until it has been delivered. In general, all remaining activities necessary to finalize the activity should be performed while in this state. This could involve the creation of a release branch, the deployment in production, documentation updates, etc.

12.10.2Responsible Role

Project Manager.

12.10.3Inputs

None.

12.10.4Outputs

- Associated artifacts necessary to finalize the activity, such as documentation updates, configuration artifacts, etc.

12.10.5Transitions

- To **Closed**, once the Epic has been delivered.

12.10.6Tollgates

None.

12.11**State: Closed**

12.11.1Purpose

This is the final state. No more work can be done in the context of this ticket. If additional work is required, a new ticket should be created.

12.11.2Transitions

- To **Open**, to allow for errors, i.e., the ticket was closed before it was ready. This transition may be used by project administrators only.

13 **Research Project Workflow**

The Research Project workflow is presented in Figure 5.

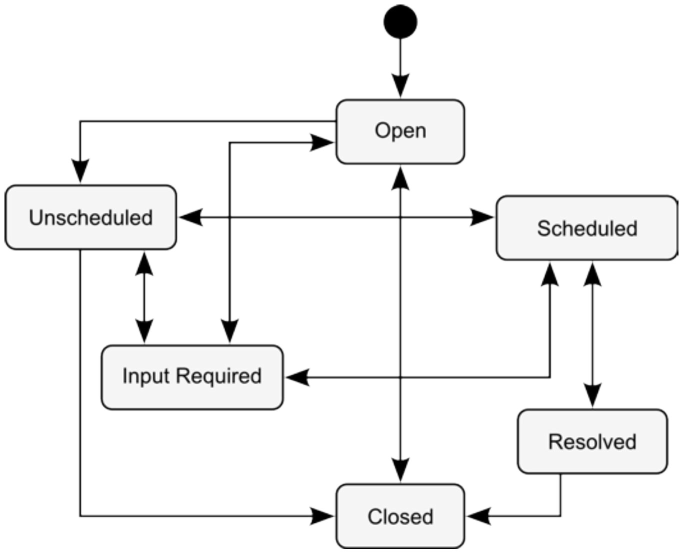


Figure 5: Research Project workflow.

13.1 State: Open

13.1.1 Purpose

This is the initial state after a new Research Project is created by Reporter. The main activity that needs to be performed while in this state is to review the goals of the project, and accept or reject the request.

13.1.2 Responsible Role

Group Lead or Component Lead.

13.1.3 Inputs

It is expected that Reporter, who creates the ticket, has provided the following inputs:

- Research Project Goals.
- Agreement with the requester of the project to actively be part of the team.
- Any additional supporting information.

13.1.4 Outputs

- Accepted or rejected decision, with accompanying explanatory comment.

13.1.5 Transitions

- To **Input Required**, if additional information is required. The ticket is assigned to the most appropriate person to provide an answer, the Expert.

- To **Unscheduled**, if the Research Project is accepted.
- To **Closed**, if the Research Project is rejected.

13.1.6 Tollgates

- **Sufficiency of stated goals and supporting information.** The Group Lead evaluates if the specified goals and additional information are sufficient to initiate the Research Project.

13.2 State: Unscheduled

13.2.1 Purpose

This state indicates that the ticket has been accepted, but hasn't been scheduled yet. In Agile methodologies, the collection of tickets in this state is referred as the **"backlog"**. The main operation necessary in this state is for Developer to perform the effort estimation and schedule the task.

Note that in the case of Research Project, it may not be possible to perform a very meaningful effort estimation. The ticket is time-boxed, allocating an amount of time judged as reasonable for the stated goals.

13.2.2 Responsible Role

Developer.

13.2.3 Inputs

- Research Project Goals.
- Additional supporting information and references.

13.2.4 Outputs

- Effort estimation.
- Schedule, specifying a start and end date.

13.2.5 Transitions

- To **Closed**, if Developer finds out that no work is actually needed.
- To **Scheduled**, when a start and end date has been assigned for the ticket.
- To **Input Required**, if Developer finds that additional information or clarifications are needed in order to estimate the effort.

13.2.6 Tollgates

None.

13.3 State: Input Required

13.3.1 Purpose

This state is used to indicate that the ticket requires additional information.

13.3.2 Responsible Role

Expert.

13.3.3 Inputs

- A clear statement specifying what information is required, and why it's necessary to continue with the process.

13.3.4 Outputs

- The missing information.

13.3.5 Transitions

- Back to **Open**, after providing the requested information.
- Back to **Unscheduled**, after providing the requested information.
- Back to **Scheduled**, after providing the requested information.

13.3.6 Tollgates

None.

13.4 State: Scheduled

13.4.1 Purpose

This is the state where the Research Project is executed.

13.4.2 Responsible Role

Developer.

13.4.3 Inputs

- None.

13.4.4 Outputs

- A report, documenting the work performed, and what was found.

13.4.5 Transitions

- To **Input Required**, if during the implementation the developer finds out that additional information is required. The ticket is assigned to Expert.
- To **Unscheduled**, if the Research Project needs to be stopped (because of the submission of a Blocker bug, for example). This transition is also necessary to deal with process errors (i.e., the ticket was moved to Scheduled by mistake).
- To **Resolved**, once the Research Project has been completed or the associated time-box for it has expired.

13.4.6 Tollgates

None.

13.5 State: Resolved

13.5.1 Purpose

In the Resolved state, the report is reviewed by Reporter and interested parties.

13.5.2 Responsible Role

Reporter.

13.5.3 Inputs

- Research Project report.

13.5.4 Outputs

- Feedback about the Research Project report, specifying if any additional work is needed.

13.5.5 Transitions

- To **Closed**, if the time allocated in the time-box has already been used, or if Reported judges that the goals of the Research Project have been accomplished.
- Back to **Scheduled**, if additional work is needed, **and there is time available in the time-box**. This transition is also provided to correct errors, i.e., the ticket was moved to Resolved before it was ready.

13.5.6 Tollgates

None.

13.6 **State: Closed**

13.6.1 Purpose

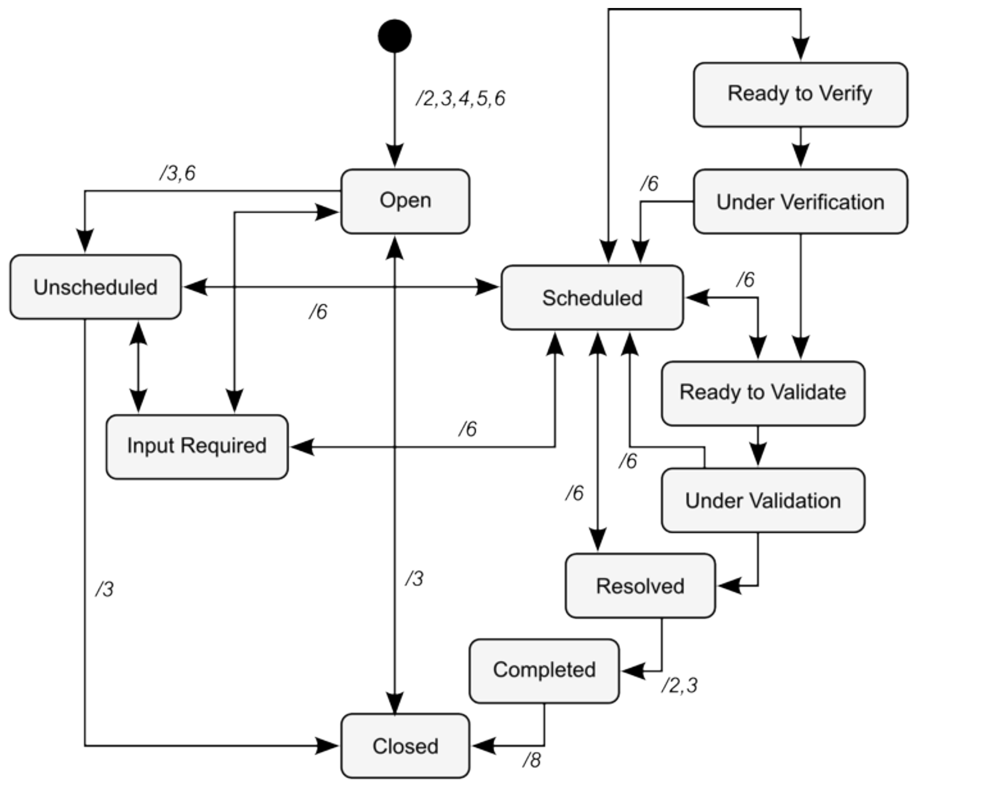
This is the final state. No more work can be done in the context of this ticket. If additional work is required, a new ticket should be created.

13.6.2 Transitions

- To **Open**, to allow for errors, i.e., the ticket was closed before it was ready.

14 Notifications

One aspect about the implementation of the workflows that is worthwhile to define is the manner in which the participant roles get notified of ticket changes and transitions. Figure 6 presents a possible notification scheme, discussed with CASA and the SSA teams. Note that only the notifications that don't fall under the "General Notification Rules" are shown explicitly in the diagram.



Participant Roles:
1. Reporter
2. Project Scientist
3. Group Lead
4. Component Lead
5. Component Scientist
6. Project Manager
7. Sofware Architect
8. Developer
9. Test Engineer
10. Validator
11. Acceptance Body
12. QA
13. Expert

General Notification Rules:
A. Ticket watchers are notified in all transitions.
B. Reporter is notified in all transitions.
C. The ticket assignee is notified in all transitions until the ticket is assigned to somebody else.

Figure 6: Workflow notification scheme. This diagram defines the roles that should be notified on different transitions.

15 **Artifacts**

This section describes the artifacts that should be provided or produced in several stages of the workflows, and defines templates for each one of them. Table 1 summarizes the artifacts required for each type of request.

Artifact/Type	**Bug**	**Feature**	**Task**	**Epic**	**Research Proposal**

Requirement Spec.	(Bug description)	x	x	x	(Goals)
Project Plan				x	x
Test Plan	x	x	x	x	
Test Report	x	x	x	x	(Report)
Validation Plan		x		x	
Validation Report	x	x		x	

Table 1: Artifacts required per request type.

15.1 Requirement Specification Document

Requirement elicitation and analysis is a complex activity, involving creative and innovative thinking, effective communications with stakeholders, and a systematic approach for elaborating and writing requirements. This document doesn't prescribe a particular process or methodology for requirement elicitation and analysis, but it does require that *a process* be defined and *a Software Requirement Specification document* be produced by the group and its stakeholders. This document should be reviewed and accepted for quality while in the Open workflow state, before moving forward with the next development activities. The Group Leader (or his delegate), should review the requirements for understandability, testability, and other necessary qualities before accepting them.

The type of process adopted by each group will vary with respect to the nature of the task or project, the frequency and quality of communications with stakeholders, and the evaluated risks involved in the effort. For simple features and bugs, a few sentences could be adequate. For other, more complex requests, a more formal document will need to be written. These factors should be judged carefully by the Group Leader and DMSD management, and help and guidance should be solicited to the PMD and Systems Engineering teams when needed.

An **example** template for a software requirements specification is presented below. This can be customized to fit the specific needs of a project. Other templates are available from past requirement elicitation efforts performed by development groups and from the extensive literature that exists about the subject.

```
1. Introduction
 1.1 Purpose
 1.2 Document conventions
 1.3 Project scope
 1.4 References
2. Overall description
 2.1 Product perspective
 2.2 User classes and characteristics
 2.3 Operating environment
 2.4 Design and implementation constraints
 2.5 Assumptions and dependencies
3. System features
 3.x System feature X
        3.x.1 Description
        3.x.2 Functional requirements
```

```
4. Data requirements
 4.1 Logical data model
 4.2 Data dictionary
 4.3 Reports
 4.4 Data acquisition, integrity, retention, and disposal
5. External interface requirements
 5.1 User interfaces
 5.2 Software interfaces
 5.3 Hardware interfaces
 5.4 Communication interfaces
6. Quality attributes
 6.1 Usability
 6.2 Performance
 6.3 Security
 6.4 Safety
 6.x [others]
7. Other requirements
Appendix A: Glossary
Appendix B: Analysis models
```

15.2 Project Plan Template

```
1. Short Description.
2. Reporters and stakeholders.
3. Current Status.
4. Predicted Duration.
5. Project Charter. [Including clear statement of purpose and scope.]
6. Prerequisites.
7. Requirements.
 7.1 Must have.
 7.2 Should have.
 7.3 Could have.
 7.4 Will not have.
8. Implementation Plan.
[Includes schedule of work activities specifying objective milestones.]
[Include resource allocations.]
9. Resources. [Includes resources in addition to the group software personnel (i.e.,
PMD, Architect, Test group.]
9. Risk Management Plan.
10. Impact on Computing Resources.
```

15.3 Test Plan Template

```
1. Short Description. [A sentence describing the test, e.g., "Simple archive query",
or "ALMA interferometric pipeline execution".]
2. References. [Reference information, including JIRA ticket #s.]
3. Purpose. [What is to be tested?]
4. Required Setup and Environment. [Required OS, database setups, special hardware,
etc.]
5. Preconditions.
```

```
6. Inputs.
7. Test instructions.
8. Postconditions.
9. Alternative cases.
```

15.4 Test Report Template

```
1. Environment details. [Anything relevant about the environment used for testing.]
2. Result. [Check one of the following: "Passed", "Not passed", or "Passed with
conditions".]
3. Conditions. [If "Passed with conditions", explain here what are these conditions.]
4. Additional information. [Stacktraces, core dumps, logs, etc.]
```

15.5 Validation Plan Template

```
1. Short Description. [A sentence describing the test, e.g., "Simple archive query",
or "ALMA interferometric pipeline execution".]
2. References. [Reference information, including JIRA ticket #s.]
3. Purpose. [What is to be tested?]
4. Required Setup and Environment. [Required OS, database setups, special hardware,
etc.]
5. Preconditions.
6. Inputs.
7. Test instructions.
8. Postconditions.
9. Alternative cases.
```

15.6 Validation Report Template

```
1. Environment details. [Anything relevant about the environment used for testing.]
2. Result. [Check one of the following: "Passed", "Not passed", or "Passed with
conditions".]
3. Conditions. [If "Passed with conditions", explain here what are these conditions.]
4. Additional information. [Stacktraces, core dumps, logs, etc.]
5. Documentation evaluation.
```

15.7 Release Acceptance Plan Template

```
1. Introduction.
 1.1 Purpose.
 1.2 References.
 1.3 Glossary.
2. Release Overview.
 2.1 Expected features.
3. Preconditions.
[Enumerate conditions that are necessary to start the validation tests, e.g., database
migrated to a new schema, special release installed, etc.]
4. Test period.
[When the validation tests are going to be performed?]
5. Resources.
[What resources are necessary to perform the tests? Cite human, computing and
telescope resources.]
6. Validation Test procedures.
[Reference Validation Test IDs.]
```

15.8 Release Acceptance Report Template

```
1. Introduction.
 1.1 Purpose.
 1.2 Acceptance Review Committee
 1.3 References.
 1.4 Glossary.
2. Executive Summary.
 2.1 Decision. [Accepted/Not accepted, accompanied by the reasons behind the 
decision.]
3. Release Overview.
 3.1 Delivered features.
4. General problems.
5. Feature problems.
6. Other comments.
```

16 Colocation of team staff

DMSD recognizes that frequent interaction between members of a software team promotes the exchange of ideas and minimizes the existence of functional silos. DMS is committed to working with the business and administration department within NRAO to

- facilitate colocation of staff within the same software development team, to the extent possible;
- advocate for the establishment of “shared space,” or unreserved meeting areas that facilitate collaboration, exchange of ideas, group problem solving, and serendipitous encounters between team members;
- advocate for the establishment of DMS “information radiators,” or public displays that communicate team status to people walking by.

The effectiveness of the department’s efforts to achieve the above goals will be reviewed annually by each software team, during a retrospective or other meeting format, and the results communicated to the software division head.

17 Appendix A: Implementation Notes

DMSD has decided to support the processes described in this document by means of the Atlassian suite. The following applications will be purchased, installed, and used by each DMSD development group:

- **JIRA**: For change and bug tracking. Workflows and fields can be customized, facilitating the implementation of the processes described in this document.
- **Confluence**: A document management system, similar to a Wiki, but well integrated with JIRA and other Atlassian applications. Its Calendar plug-in allows to implement the Master Schedule (see Section 7). It also provides a medium to manage document templates and their instantiations.
- **Bitbucket Server**: Provides a host repository for Git. Allows pull-requests, which some groups will use as part of their SCM processes.
- **Bamboo**: A continuous integration server.

18 Appendix B: Group Management Document Template

Besides the artifacts defined in Section 15, which are specific for each request or release, each DMSD development group will also maintain a Group Management Document, that will be updated annually and reviewed by DMSD management and other stakeholders. This document aims to communicate general information about the group and its medium-term (~1 year) plan.

```
1. Group goals and scope.
2. Group organization.
3. Software development process.
       [Scheduling policy.]
4. Software configuration management.
       [Branching strategy.]
5. Personnel and skillsets.
6. Personnel integration plan.
7. Identification of stakeholders.
8. Brief system description and external interfaces.
9. Planned development activities.
       [Release plan.]
10. Planned maintenance and evolutionary activities.
11. Development resource allocations.
12. Maintenance resource allocations.
13. Risk analysis, mitigation strategies.
```

The planned development activities in particular should be written for both internal and external audiences and should include preconditions, or events that must occur for the activity to be successful (third-party software availability; research project approval, etc.).