

PROCESS DOCUMENTATION

“Release 1.0”

Ready Wireless – Outsource Process

Prepared by: **IT Department**

*Date: 3/17*/2014

Contents

[Document Information and Approvals 3](#_Toc382811488)

[Version Control 3](#_Toc382811489)

[Approvals 3](#_Toc382811490)

[Overview 4](#_Toc382811491)

[Purpose 4](#_Toc382811492)

[Assumptions 4](#_Toc382811493)

[Process Definition 4](#_Toc382811494)

[Process Overview 4](#_Toc382811495)

[Development Process Overview 4](#_Toc382811496)

[Sprint Process 4](#_Toc382811497)

[Task Workflow 5](#_Toc382811498)

[Release Process 6](#_Toc382811499)

[Source Code Overview 7](#_Toc382811500)

[Reference A 8](#_Toc382811501)

[Reference B 9](#_Toc382811502)

# 

# Document Information and Approvals

### Version Control

|  |  |
| --- | --- |
| Version History | |
| **Version #** | **Date** | | **Revised By** | **Reason for change** |
| 1.0.0 | 03/17/2014 | | Scott Bonen | Initial draft |

### Approvals

This document is intended for external distribution and represents the official Ready Wireless Developer Standards for third party developers.

Following approval of this document, requirement changes will be governed by the project’s change management process, including impact analysis, appropriate reviews and approvals.

|  |  |
| --- | --- |
| Document Approvals | |
| **Approver Name** | **Title/Role** | | **Signature/Electronic Approval** | **Date** |
| Mike Schmidt | VP, Technology | |  |  |

# Overview

## Purpose

This document defines the workflow and processes that are utilized by outsource resources for ReadyWireless IT.

## Assumptions

* Development environment is setup and functional. Reference document “Environment Overview”
* Developer has access to ReadyWireless JIRA and BitBucket cloud services
* Developer has general understanding of JIRA Agile workflow and Git source code control

# Process Definition

ReadyWireless utilizes Atlassian JIRA cloud services (<https://readywireless.atlassian.net>) for managing development processes. The following outlines the general overview of these processes.

## Process Overview

1. New requirements are evaluated daily by the IT Department Manager for completeness and accuracy.
   1. New requirements not following process or incomplete are returned to requester ASAP.
   2. Priorities are assigned by Department Manager as they become known.
2. Weekly IT task and priority meetings are held every Tuesday afternoon.
   1. Meetings are broken into functional groups and include management and team leads.
   2. Sprints are created
   3. Tasks are reviewed for completeness and assigned priorities
      1. Note: Any task with a work estimate of more than 60 minutes will be broken down into multiple tasks. No task should have an estimate greater than 60 minutes.
   4. Release stories are created for upcoming release
   5. Tasks will be bucketed into Sprints based on priorities
   6. Tasks requiring additional definition will also be discussed
3. Weekly meetings will be held between IT Management, Outsource resources, and Project Leads. Date/Time of meetings will be determined based on resource location and availability.
4. Outsource resources will work tasks contained within the defined Sprint and follow the process as outlined in the “Sprint Process” section.

## Development Process Overview

1. Check regularly for available work items either assigned or unassigned with status “TO DO”.
2. If not assigned, assign work item to yourself and drag to “IN PROGRESS”.
   1. Compile-able code must be checked in by outsourced developers at the end of the work day each day. Code may not be functional, but must compile.
   2. Code will be reviewed daily and issues noted for the developer before the next shift.
3. During the coding phase test cases must be created to support/prove accurate implementation.
4. Thoroughly test changes for completeness using pass/fail test scenarios created.
5. Once development/testing are completed, move the item to “CODE REVIEW” and request a code pull.
6. If code approved move to QA if comments contain “Must QA”, otherwise move to “DONE”.
   1. If outsourced developer always move completed tasks to QA for final testing.
7. If code is not approved, correct the issues identified and repeat step 5 & 6.

## Sprint Process

The following outlines the IT sprint process.

1. Sprint planning occurs every Tuesday during the planning meeting.
   1. Sprints are named by project. Example: Platform Sprint 1 (Number is incremental)
2. Sprint Cycle
   1. Start dates always fall on a Wednesday (Follows Tuesday planning)
   2. Sprints follow a 1 week cycle with an end date always landing on Tuesday
   3. Unfinished tasks will be moved into the next Sprint
3. Tasks will be prioritized weekly
   1. High priority tasks can be added to current sprints in progress. This will affect the sprint scope and may result in not all tasks being completed within the given sprint.
   2. All other tasks will either be added to upcoming scheduled sprints or left in the backlog for the next planning meeting.
4. Release Sprint – This is a normal task sprint but has two additional stories for building a production release. This only applies to prior completed sprints and not the sprint containing the release stories.
   1. Story for QA/UAT testing
      1. Any QA/UAT issues found are assigned to this story
   2. Story for Release
      1. Default list of tasks required to build a release
   3. If release is delayed due to issues, (bugs, scope creep, etc) the release stories can fall into the next Sprint or be canceled.
   4. Reference section “Release Process” for more details

## Task Workflow

The flow of a task is defined and enforced by the JIRA workflow for all projects. Outsource resources will work tasks from the active sprint. Scheduling functionality is not available. Management or Project Leads may assign tasks directly to development resources. All unassigned tasks are available to be worked as assigned tasks are completed.

Workflow Detail:

* TO DO: This state signifies an available task that can be assigned to a resource
  + Transitions:
    - “To Do” -> “In Progress”
  + Work Items:
    - Create new task branch under the feature directory
      * Branch name based on the task name. IE PLAT-35
      * Branches are created from the develop repository under the feature directory
* IN PROGRESS: This state signifies a task that is currently being worked
  + Transitions:
    - “In Progress” -> “Needs Feedback”
    - “In Progress” -> “Code Review”
  + Work Items:
    - All source code commits need to contain the project identifier and task number. This ensures that the commit is properly attached to the work task. Example identifier -> PLAT-25 or CRM-18
* NEEDS FEEDBACK: This state signifies a task that is pending additional information and/or feedback
  + Transitions:
    - “Needs Feedback” -> “In Progress”
* CODE REVIEW: This state signifies a task that has been completed and is pending review of the work accomplished
  + Transitions:
    - “Code Review” -> “In Progress” (Rejected)
    - “Code Review” -> “Approved” (Accepted)
* APPROVED: This state signifies a task that has been approved and is ready to be either moved to QA or Done
  + Transitions:
    - “Approved” -> “Done” (Task marked as not requiring QA testing)
    - “Approved” -> “Pending QA” (Task requires QA testing)
  + Work Items:
    - Work task branch is closed (removed)
    - Update/Set work task time tracking
* QA: This state signifies a task that is in the QA process
  + Transitions:
    - “QA Pending” -> “QA In-Progress”
    - “QA In-Progress” -> “Done”
* DONE: This state signifies a task that all work has been completed and is ready for release. Note: Project Leads can move a task from any state to this state.
  + Transitions:
    - “Done” -> “Release”
* Additional Notes:
  + The workflow enforces transition permissions. Not all users will be able to move tasks between different transition points

## Release Process

This section outlines the general process flow for production releases. Releases occur on the first and third Tuesday of each month at 6am CST. The dates and time may fluctuate based on business needs and/or IT adjustments. Reference Section B for Gitflow Release process.

* Hot fixes (Production bug resolutions) will be handled via the Hotfix Gitflow process and will result in a bug fix version identifier.
  + Based on severity this may result in an out of cycle production release.
* Minor JIRA Stories/Tasks follow the process as outlined in the source code overview section. These stories/tasks will be merged directly into the develop branch.
* EPICs will require a Gitflow feature branch and all associated stories/tasks fall within this feature branch.
  + When all stories/tasks associated to the epic have been completed Bamboo will generate a QA build for testing.
  + Once QA testing has been completed and accepted the epic branch (feature) will be finalized with all changes being merged into the develop branch and making the epic available for the next production release.
* Release branches will be generated by the Gitflow release process. This will generate a release branch based on the current state of the develop branch for the repository.
  + Release branch naming will define the release version. Release versions are defined as follows: R<major>.<minor>.<hotfix>
    - Major -> Last 2 digit of current year. IE 14
    - Minor -> Starts at 0 and is incrementally updated for each release. Start back at 0 if major is incremented.
    - Hotfix -> Production bug fix releases. Starts at 0 and is incrementally update for each minor release. Start back at 0 if minor is incremented.
  + The release will include all minor and epic stories/tasks completed since the last release.
  + Creation of a release branch will result in Bamboo generating a QA instance for UAT.
  + Any issues found during QA/UAT will generate a corresponding JIRA task associated with the release story. All source code corrections will be against the release branch.
  + Upon QA/UAT acceptance and deployment to production the release branch will be finalized, resulting in the release branch being merged into develop and master branches.

## Source Code Overview

Source code for all projects reside on BitBucket (<https://bitbucket.org/readywireless>) cloud servers. This is a closed repository with specific permissions granted to individual users. This section outlines source code control for outside resources.

* JIRA tasks requiring source code changes require a branch be created for the specific task. This is easily accomplished by selecting the “Create Branch” on the JIRA Task.
  + Branches will be named based on the JIRA task ticket name. For example PLAT-25
  + Branch locations will be contained within one of two locations
    - If standalone task the branch falls under the feature directory. When creating the branch the full name will be “feature/PLAT-25”
    - If task is part of a larger task group the branch will be created under an existing feature. Example: feature/Wildfly/PLAT-25
    - Note: The attached “Task Definition Form” will specify location of task branch
* All code modifications for the task will be committed within the created task branch
  + Commit comments must contain the JIRA ticket so the commit can be attached to the JIRA tasks.
* When the task is ready to be moved from the “In Progress” state to “Code Review” state and all code changes have been committed, the developer will create a pull request for the task branch. This will generate a code review request.
  + Prior to generating the pull request the JIRA task needs to be moved into the “Code Review” state
  + All pull requests will merge the task branch into the “develop” or specific “feature” branch. Never to “master”
  + Assign the pull request to the project lead for review.
  + If there are no additional code changes for the task check the “remove branch”. Note: Interim code reviews can be requested. Provide in comment that this is an interim code review request.
* Project leads will review the pull request and either accept or decline the request.
  + If declined, project lead will move the JIRA task back to “In Progress” and will provide reason for decline
  + If accepted, project lead will approve the pull request and move the JIRA task to “Approved”.
* Developer finalize tasks
  + Complete pull request by merging the code into the “develop” or “feature” branch
  + Verify task branch has been removed from source control
  + Update time estimates
  + Additional comments as needed
  + Move the JIRA Task to either “Pending QA” or “Done”

# Reference A

The following is an example of a requirement document that must be attached with each Jira task.

**Task Definition Form**

**Java  PHP**

|  |  |
| --- | --- |
| **Project Name** | PlatformBase (Name of Java/PHP project) |
| **Jira Task** | PLAT-25 |
| **Class Name(s)**  **Source Name(s)** | OrderBase.java (List of Java classes or PHP source code that will be modified) |
| **Method Name(s)** | getAdhocQuote() (Defines specific method name being modified/created) |
| **Repository Branch Location** | feature/PLAT-25 (Signifies task branch location) |

**Business Need/Workflow** (Problem statement and Use Case)

Due to tax code changes, sales tax needs to be charge against shipping costs. During the order process a quote is required for showing the consumer total cost of purchase.

**Technical Description of Change:**

getAdhocQuote() should return shipping charges.

**Requirements:**

getAdhocQuote() calls the internal private method addLineItems(). Within the method addLineItems() there is a “IF” statement that checks if the line item is of type Shipping and if so skip the line. This if needs to be modified to no longer skip “Shipping” line items. Remove the check from the IF

**Reference Documentation:** (Provide additional documentation as needed)

Examples:

* Database Modifications
* Third Party Libraries
* Object Model
* MVC Layout

# Reference B

The following outlines the ReadyWireless Gitflow Process

