Customer Onboarding Guide

Essentials for Product Owners

Introduction

First off, welcome! We are thrilled to be working with you! We have put together this guide to help you make the most of your experience in working with Radial Development Group. This guide will define some common industry terms, roles, and team expectations that we have found to be critical in successful launches. This guide specifically outlines the processes and tools Radial uses to manage the development of your product. We hope that you'll take a few minutes to review this information and of course, if you have any questions - just ask!

Roles and responsibilities

Product Owner

The Product Owner is responsible for defining requirements and accepting completed work. Before a sprint, the Product Owner should prioritize major features and select one to focus on in accordance with the business goals of the project. The feature should be described in detail, sometimes with a designer (see Specifications and Stories). During the sprint planning meeting, the Product Owner should describe the business goals and provide user story specifications. Once a sprint is in progress, the Product Owner should be available to answer questions about the specifications. The Product Owner is also responsible for final testing and acceptance of completed work which can be done as Developers complete items.

Developer Lead

The Developer Lead is the primary point of contact coordinating the Development team and the Product Owner and is responsible for ensuring timely communication occurs. The Developer Lead should notify management as soon as possible if more resources are needed or if progress is otherwise blocked. They are also the final authority for communication and code standards. Most importantly, the Developer Lead is responsible for delivery.

Designer

The primary role of a Designer is to find the best way of achieving a specific business goal. There are many levels of design. User Experience (UX) design focuses on the overall flow of the application as well as any offline interactions that may be involved. User Interface (UI) design is about creating a consistent design language for elements such as buttons and panels. Graphic Design or Illustration is the creation of specific graphics for use in the final product. Designers will often use tools such as flow diagrams, wireframes, and mockups to think through and communicate the design.

Developer

Developers are responsible for selecting appropriate frameworks and technologies to meet the business goals of the project. During sprint planning, developers will “weight” feature stories according to how much work they are to implement and break them into smaller steps if necessary. Developers are responsible for maintaining code quality through automated tests and code review. Developers should take into account described goals while working on implementing specifications and suggest cheaper/better alternatives when applicable.

Deployment and infrastructure

Reliable availability for your application is critical. To achieve this, we use duplicate servers so that new code can be tested and reliably shipped to your users.

Environments: Staging, Stable, Production

- **Staging** is for development testing. Once a feature is completed by a developer it gets deployed to this server for acceptance testing. Please note: data and settings may differ from the production server. This server may also run slower than the production server due to lower cost of infrastructure.
- **Stable** is generally a duplicate of the production code. Not all projects will have this server in the deployment flow but it is useful for high-rate development. It is useful as a dry run for deploying a sprint to production. It also serves as a place to test “hotfixes” without disrupting main feature development. Data and settings may also differ from the production server but often data will be copied over from production for testing.
- **Production** is the server your users will access.

When testing and reporting bugs it is important to bear in mind which server you are using. Typically you can look in the URL bar in your browser to see if you are on the staging or stable servers.

Continuous integration

As much as possible, deployment of new code should be routine and automated. This ensures that completed features can be shipped to users quickly and valuable developer time is not wasted at the eleventh hour. In general, changes made to the codebase should automatically be deployed to staging within minutes. Once on staging (or stable if applicable), changes can be “promoted” to production using an automated deployment command.

Key services

Typically, production applications rely on additional services besides the main application server. Here are some common examples:

- **Media storage** - dynamic content, for example user uploaded photos, is stored on a storage service such as Amazon S3.
- **CDN (Content Delivery Network)** - proxy service for handling high loads for frequently requested assets such as Cloudflare or Akamai.
- **Transactional email service** - for reliably sending emails to users such as SendGrid.
- **Error reporting and diagnostics** - for logging and diagnosing operational issues such as Rollbar and New Relic.

Scaling up

We typically use Platform-as-a-Service providers such as Heroku and Netlify to host the primary application server. These have the advantage of being really quick to get started but are configured to scale up quickly if needed. Because the application is architected to run on these platforms it can be migrated to larger customer infrastructure relatively cheaply and quickly by a DevOps company.

Specifications and stories

High quality specifications improve development efficiency and quality. There are three categories of work requirements:

Features (stories)

Features are new functionality with an associated business value or outcome. Often a feature will be broken down into a series of smaller features to define the specific functionality to be built. An effective way of communicating requirements is to use the “user story” format which connects a user persona with the feature to be developed and the outcome that user desires.

Example: **As a** *paid member* **I want to** *upgrade my subscription* **so that** *I can access more content.*

‘As a’: refers to the person performing the action such as ‘admin’, ‘site-owner’, ‘paid member’, or ‘site-visitor’. You can use specific roles you’ve defined within your organization (e.g. accountants, tenants, editors).

‘I want to’: the action the user will perform.

‘So that’: The most important part of a story. This is the reason the user is performing the action. By filling this in, you are providing the development team with valuable context around your story.

If this seems a bit overwhelming to start, don’t worry! Stories are so critical to product development that we will work together to make sure we have good stories before each and every sprint.

Defects (bugs)

Defects are errors discovered with existing functionality. If an error message appears it is helpful to screenshot the message along with a description of what action led to the error as well as what user account you might be logged into. If an error message does not appear but something happens that is incorrect (or nothing happens) describe the expected behavior and the current behavior.

Defects have different levels of severity depending on if they are discovered in production, how many users they affect, and how big the impact is. Depending on the severity level they may be addressed immediately (with a hotfix on production), included in the next release, or deferred until some future time.

Chores (tasks)

Sometimes there is work to be done that does not result in new functionality but is not the result of a defect. Examples: most kinds of text changes or dependency upgrades.

Sprints and delivery states

Feature work is often done in a fixed time period called a “sprint”. This helps to routinely deliver incremental improvement.

Sprint cadence

A sprint begins with a Sprint Planning Meeting. During the meeting the Product Manager presents prioritized feature stories to the development team. The team asks questions until they are satisfied they understand the requirements and “points” the stories indicating the estimated time required. Based on the estimates, the Product Manager may elect to defer some functionality in order to ensure that the sprint can be assured of delivering value.

During the sprint, developers begin work on stories one at a time. When the work on a story is completed it is reviewed by another developer and merged into the codebase then “delivered” to staging.

The product owner should then verify that the feature works as intended on staging keeping in mind that some things may differ from the comps that will be implemented in future stories. If the feature meets spec then the story should be marked as accepted. If not, it should be rejected with a note as to the defect.

At the conclusion of a sprint, all work should be accepted and the iteration can be promoted to production. Often there will be a retrospective to document problems encountered and propose solutions going forward.

Tools (Tracker)

Many projects at Radial are managed through an online project management tool known as Pivotal Tracker. Tracker is your primary communication tool with the development team and your window into how your product is being developed. From within the application, you will be able to see the progress of your project and communicate about the status of work items. If you have not received an invitation to Pivotal Tracker, please contact your Dev lead.

The icebox (or “Someday”) section is for future work that has not yet been prioritized but still needs to be documented for future reference. The backlog is for prioritized work that has been scheduled for a sprint.

Important: Once a sprint has begun, additional work should be added to **subsequent** sprints instead of the current one. If an issues arises that demands immediate attention mid-sprint, please work with your Dev Lead in getting it resolved. Often times, these issues can be resolved without disrupting the flow and productivity of a scheduled sprint.

Once a work item has been accepted it becomes complete.

Communication about specific work items should be documented in the comment section of that item as emails are often hard to reference later when the work actually begins. Special attention

should be given to any stories tagged “blocked” as this indicates that the developer is unable to continue to work on the item without additional information.

Testing and acceptance

Accepting stories is a critical responsibility of the Product Owner as it provides valuable feedback to the developers and prevents errors from compounding.

When testing it is important to keep in mind what server environment (staging or production) you are on and what user account you are logged into. It is also helpful to reference the story being tested for acceptance as some features you may be expecting may be implemented in a future story. If a feature does not appear to be functional or functions differently than described, reject with a note describing the error. If a feature works as expected but is not ready for release due to qualitative factors, reject with a note describing specific defects encountered. You may elect to accept a story despite some imperfections to get it released sooner. In this case, create a defect documenting the problems in the next sprint.

Important: Deploying features to production is dependent on (and therefore can be delayed by) the Product Owner rate of accepting stories.

Appendix 1: Technology stacks

What is a stack?

Front-end/back-end?

Servers and services