radial
DEVELOPMENT GROUP

Client Onboarding Guide

Essentials for Product Owners

Introduction

On behalf of the entire Radial team, we're excited to start our project together! We developed this guide over years of client work as part of our quest to make working with our team the most delightful, efficient experience possible.

The purpose of this guide is to define common industry terms, project roles, and team expectations we have found critical to successful launches. This guide specifically outlines the processes and tools Radial uses to manage the development of your product. Please take a few minutes to review in full and let us know if you have any questions at all.

We can't wait to get started!!

Roles and responsibilities

Product Owner

The Product Owner is the person on your team who is responsible for communicating with Radial and defining project requirements. The Product Owner is also responsible for final testing and acceptance of completed work which can be done as Developers complete items.

The Product Owner's primary point of contact at Radial is the Developer Lead. Before a **sprint** (*a sprint is a set period of time during which specific work has to be completed and made ready for review*), the Product Owner will prioritize major features and select one to focus on in accordance with the business goals of the project. During the sprint planning meeting, the Product Owner should describe the business goals and provide user story specifications. As part of this process, we'll help the Product Owner describe the feature in detail, sometimes with a designer (see Specifications and Stories). Once a sprint is in progress, the Product Owner should be available to answer questions about the specifications.

Developer Lead

Responsible for ensuring timely communication and delivery at every stage of the project, the Developer Lead is the primary point of contact on Radial's side. The Developer Lead will communicate directly with the Product Owner and will notify Radial management as soon as possible if more resources are needed or if progress is otherwise blocked. The Developer Lead is also the final authority for communication and code standards.

Designer

Designers report to the Developer Lead. A Designer's primary role is to find the most effective 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 report to the Developer Lead and are responsible for selecting appropriate frameworks and technologies to meet the business goals of the project. During sprint planning, developers will "weight" 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 take into account described goals while 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 the **staging serve**r for acceptance testing. **Please note**: data

and settings on the **staging server** 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 a **stable server** in their 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** (*software updates designed to fix bugs or security holes*) 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. Once code hits the production server, it's live.

The **Staging Server** is the proof print, the **Stable Server** is the reviewer print run, and the **Production Server** is the prints that go to bookstores.

It's important to bear in mind which server you are using when testing and reporting bugs. We'll provide you with all sever locations and, 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 will be routine and automated. This ensures 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 and 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 as necessary by a DevOps company.

Specifications and Stories

Clear, detailed specifications improve the efficiency and quality of our development efforts. 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 and their desired outcome with the feature to be developed.

> 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 critical to product development and we'll work together to make sure we have good stories before each and every sprint.

Defects (Bugs)

Defects, or **Bugs,** 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 which 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 as in-depth as possible.

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 the **production server**), included in the next release, or deferred until an agreed upon future date.

Chores (Tasks)

Sometimes there is work to be done that does not result in new functionality but is also not the result of a defect. Examples of this include most kinds of text changes and code library upgrades. These usually do not have end-user facing deliverables

Sprints and Delivery States

Feature work is often done in a fixed time period called a **sprint**. This approach ensures that Radial routinely delivers incremental improvement on your project.

Sprint Cadence

Each **sprint** begins with a **Sprint Planning Meeting**. During the meeting the **Developer Lead** presents prioritized feature stories to the development team. The team asks questions until they fully understand the feature specifications and requirements and “points” the stories indicating the estimated time required. Based on the estimates, the **Product Owner** 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** then verifies that the feature works as intended on staging keeping in mind that some things may differ from the mockups that will be implemented in future stories. If the feature meets the specifications, the story is marked as **accepted**. If not, it is marked as **rejected** with detailed notes on the defect.

When all work is accepted, the **sprint** is concluded and the iteration is 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 **Pivotal Tracker**, an online project management tool. Tracker is your primary communication tool with the development team and your window into how your product is being developed. Once in the **Pivotal Tracker** application, you can monitor the progress of your project and communicate with Radial about the status of work items.

If you have not received an invitation to **Pivotal Tracker**, please contact your **Developer Lead**.

The **icebox** 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**.

Please Note: 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 **Developer 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 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** it 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.

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

Communication

Email

We use email for all communication not directly tied to in-sprint work and items that must be received and acted upon (setting up a meeting, for example). We may not check email more than once per day but we will respond within 1 business day to any email request.

When you kick off your project, your **Developer Lead** will provide you with a team.<project>@radialdevgroup.com address to send project-related correspondence to. **Please direct all project-related communication to that address**. This approach ensures all team members are aware of changes on the project and that management is in the loop to keep things on track. Your developer lead may be out sick, on vacation, or otherwise temporarily unavailable, but someone will always respond to your request if you include your project's team address.

We prefer you keep communication about project deliverables, bugs, features or expectations separate from billing inquiries. This helps ensure that communication about those items does not become confused. Please send inquiries about bills, invoices, financial accounts or hours to ben@radialdevgroup.com or accounts@radialdevgroup.com. **Developer Leads** are not always aware of the financial arrangements of the project and may not be able to respond to questions or be able to resolve your issue.

Telephone

Telephone is a great way for communicating about complex situations. It's also ideal for having a real-time back-and-forth conversation. Communication via email, text or chat quickly becomes insufficient when discussing complicated issues and unexpected, time-sensitive situations. We encourage phone calls in these situations - please call or set-up a call if you feel email won't be an effective form of communication. We will often set up a call if we feel there is a need to discuss complex issues or believe it is the fastest way to resolve an action item or communication issue. **We believe that voice and face-to-face communication is critical to keeping projects on a solid footing.**

If you are generally available for a call, please let us know, so we can quickly resolve issues as they arise.

Instant Message/Real-Time Chat Messaging

We use https://slack.com/ for intra-team communication. This is the best option for real-time or near real-time communication with our team. We're happy to add you to our Slack workspace or set up a shared channel if you would like.

For best results, please keep project related chat in designated project channels to ensure our staff can stay in the loop of and communication about your project does not get lost in more general channels, or in private conversations.

Appendix 1: Technology stacks

What is a stack?

Front-end/back-end?

Servers and services