RDD Framework User Guide

Requirements-Driven Development Framework

A comprehensive guide for AI-assisted development with GitHub Copilot

📖 Table of Contents

Introduction to RDD Framework

The Requirements-Driven Development (RDD) framework is an innovative, structured workflow system specifically engineered to integrate seamlessly with GitHub Copilot, enhancing the development lifecycle. It offers a pragmatic, workspace-centric methodology for managing software iterations, systematically documenting requirements, and efficiently executing development tasks with advanced AI assistance. RDD is designed to bring clarity and automation to complex projects, ensuring a consistent and traceable path from requirement definition to code completion. This framework aims to maximize developer productivity while maintaining high standards of quality and alignment with project goals.

Important Note

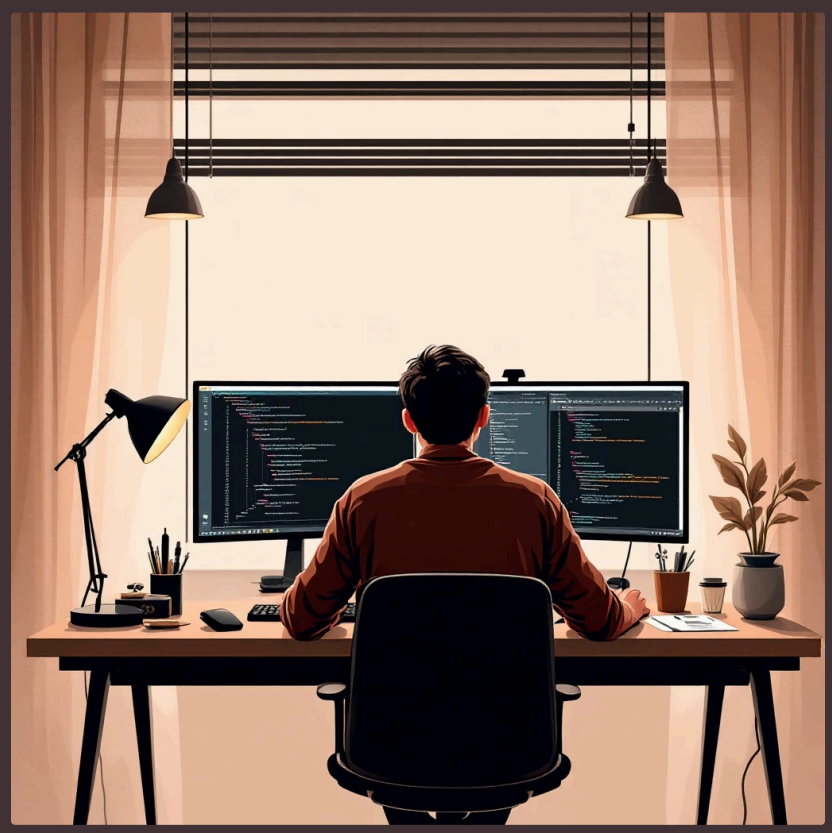
This presentation provides a distilled overview of the RDD Framework. For comprehensive details and in-depth instructions, please refer to the complete user guide document .rdd/user-guide.md.

What This Guide Covers

This comprehensive guide will equip you with the knowledge and skills to effectively leverage the RDD framework. You will learn how to: install and meticulously configure the RDD framework within your existing project environment; navigate and utilize the interactive terminal menu for granular control over development iterations; formulate and execute highly effective prompts for GitHub Copilot to generate accurate and relevant code; maintain continuous synchronization between your project documentation and development progress; and adopt a set of best practices and guidelines for optimizing AI-assisted development, ensuring robust and maintainable codebases.

Who This Guide Is For

This guide is tailored for intermediate to advanced developers who possess a solid understanding of command-line interfaces, proficiency in Git version control fundamentals, experience with code editors such as VS Code, and foundational knowledge of Python programming concepts. It assumes familiarity with software development principles and an eagerness to integrate AI tools into a structured workflow for enhanced efficiency and quality.



Installation Process

The RDD framework installation is designed to be straightforward and consistent across Windows, Linux, and macOS platforms. This section outlines the necessary prerequisites and guides you through the step-by-step installation process to get RDD up and running in your development environment.

Prerequisites

- **Python 3.7+**: Ensure Python 3.7 or a newer version is installed on your system and accessible via the `python` command in your terminal.
- **Git 2.23+**: Git must be installed for version control. You can download it from **git-scm.com**.
- **Git Repository**: Your project directory must be initialized as a Git repository. If not, navigate to your project folder in the terminal and run `git init`.

Download the RDD Framework

Visit the official **RDD GitHub Releases page**. Download the latest `rdd-v{version}.zip` file and extract its contents to a temporary location on your system. This archive contains the installer script and necessary RDD files.

Run the Installer

Open your terminal or command prompt, navigate to the directory where you extracted the RDD framework files. Execute the installer script:

- **Linux/macOS**: Run `./install.sh`
- **Windows**: Run `install.bat`

Alternatively, you can run the installer directly with Python: `python install.py`.

Configure the Installation

The installer will guide you through several configuration options:

- **Project Folder**: Select the root directory of your project where RDD should be installed.
- **Installation Mode**: Choose between a local-only installation (files are only in your project) or a GitHub remote setup (integrates with a remote GitHub repository).
- **Default Branch**: Specify the default development branch (e.g., `main`, `dev`, `master`, or a custom branch name).

Verify the Installation

After the installer completes, navigate back to your project directory in the terminal. To confirm RDD is correctly installed and accessible, run the following command:

```
python .rdd/scripts/rdd.py --version
```

You should see output similar to: `RDD Framework v{version} (Python)`, indicating a successful installation.

Note for Linux/macOS users: The framework relies on the `python` command. If this command is not found or points to an incorrect version, you might need to install `python-is-python3` on Debian/Ubuntu (`sudo apt install python-is-python3`) or create a shell alias to ensure `python` correctly invokes Python 3.7+.

Initial Setup & Configuration

After successfully installing the RDD framework, the next crucial step is to run the RDD menu to configure the framework precisely for your project's needs. This configuration step allows you to tailor various aspects of RDD's behavior, ensuring it aligns with your development workflow and repository structure.

You can launch the RDD menu through one of two methods:

1. **Using the Launcher Script**: Execute `./rdd.sh` on Linux/macOS or `rdd.bat` on Windows from your project's root directory.
2. **Direct Python Execution**: Run `python .rdd/scripts/rdd.py` from your project's root directory.

Detailed Configuration Options (Main Menu Option 5)

Once the RDD Main Menu is displayed, select option `5` to enter the Configuration submenu. Here, you'll find comprehensive settings to manage your RDD environment:

- **Update Version Numbers**: Modify the framework's version. You can choose to increment the Major, Minor, or Patch version number, aligning with semantic versioning practices.
- **Change Default Branch**: Select a different default branch for RDD operations from your existing repository branches (e.g., `main`, `dev`, `master`, or any custom branch you have created).
- **Toggle Local-Only Mode**: Switch between local-only mode (where RDD operates without connecting to a remote GitHub repository) and GitHub remote operation mode. This is useful for offline work or for projects not hosted on GitHub.

All configuration changes made through this menu are automatically saved to the `.rdd-docs/config.json` file located in your project's root. These settings persist across sessions, ensuring your preferences remain consistent and are applied automatically during subsequent RDD operations.

```
RDD Framework

Requirements-Driven
 Development

Current branch: main
Default branch: main

RDD Framework - Main Menu:
1. Create new iteration
2. Update from default
3. Complete current iteration
4. Delete merged branches
5. Configuration
9. Exit

Enter your choice (1-9):
```

Best Practice: Always review your RDD configuration after initial setup or significant project changes to ensure it matches your current development strategy. Maintaining a consistent configuration is key to a smooth RDD workflow.

Core Workflow Cycle

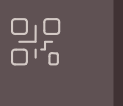
The RDD framework follows a simple, repeatable cycle for each development iteration. Understanding this workflow is essential to maximizing your productivity with AI-assisted development.

Create New Iteration

Start from default branch with empty workspace. Provide branch name and framework creates branch, initializes workspace with prompts file.

Define Work in Prompts File

Edit `.rdd-docs/work-iteration-prompts.md` and write clear, specific prompts for Copilot with full relative file paths.



Execute Prompts with Copilot

Use `/rdd.execute` command in GitHub Copilot Chat. Copilot implements each prompt and creates implementation files in workspace.

Update from Default (Optional)

Sync your branch with latest default branch changes and resolve any conflicts that arise.

Complete Iteration

Archives workspace with timestamp, commits all changes, optionally pushes to remote, and returns to default branch.

Create Pull Request

Use GitHub web interface to create PR and request code review from team members.

Merge & Cleanup

After PR is merged, delete merged branches using menu option 4 to keep repository clean.

Best Practice: Start a new chat session for each prompt execution. This keeps the context focused and reduces confusion from previous conversations.

Creating a New Iteration

Initiating a new development iteration within the RDD framework is the first critical step to leverage AI-assisted coding effectively. This process automates the setup of a dedicated feature branch and prepares your development environment, ensuring a clean and organized start for your work.

Prerequisites for Starting an Iteration

Before you can create a new iteration, ensure the following conditions are met:

- **Default Branch:** You must be currently working on the default branch (e.g., `main` or `master`) of your repository.
- **Clean Workspace:** Your local `.rdd-docs/workspace/` directory must be empty or non-existent. This ensures that previous iteration artifacts do not interfere with the new setup.

The RDD framework performs checks for these conditions. If any prerequisite is not met, the system will display an informative error message and guide you on the necessary steps to resolve the issue before proceeding.

Step-by-Step Process:

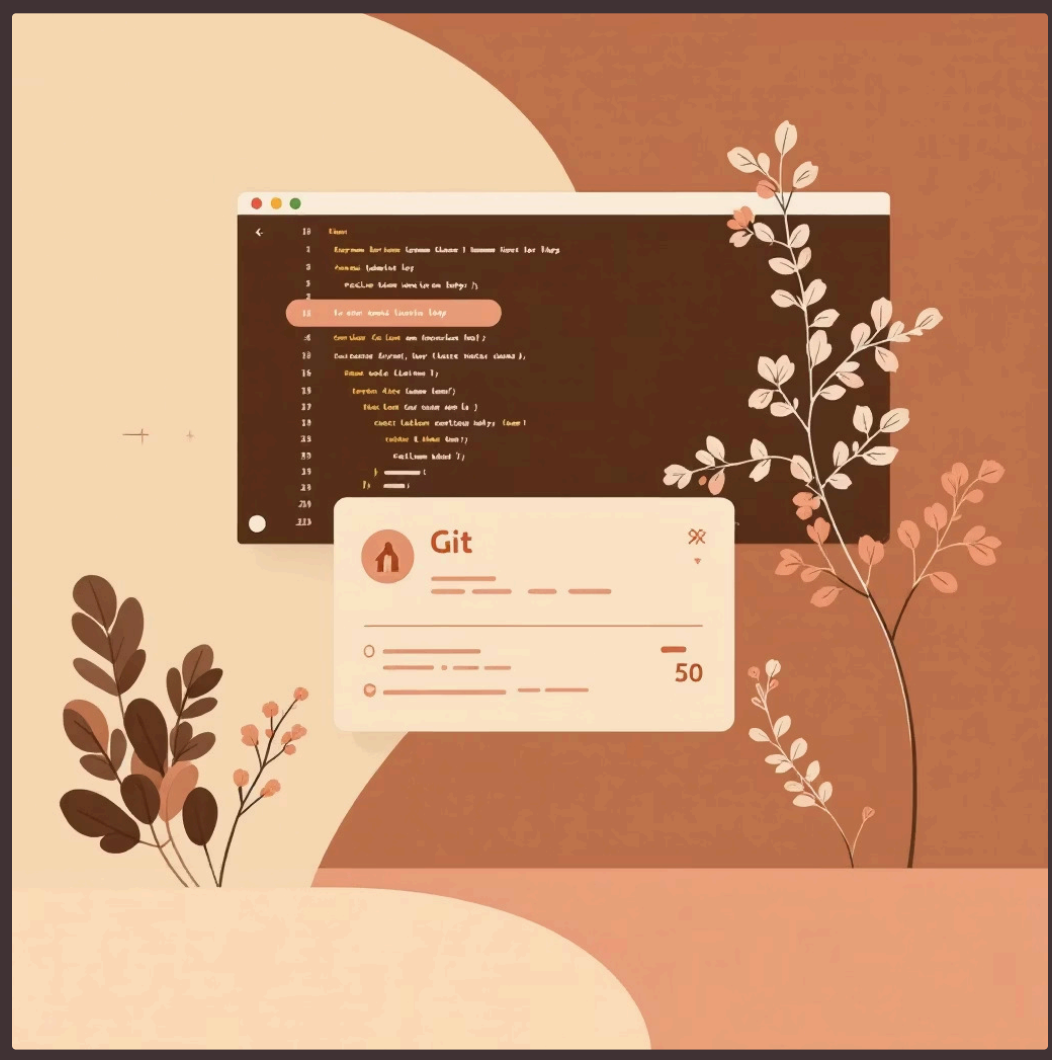
1. **Select Menu Option:** From the RDD main menu, choose option 1: "Create new iteration".
2. **Enter Branch Name:** You will be prompted to enter a descriptive name for your new feature branch (e.g., "implement user authentication feature" or "fix user login bug"). The framework automatically sanitizes and formats this input into kebab-case (e.g., `implement-user-authentication-feature` or `fix-user-login-bug`) for Git compliance.
3. **Automated Setup:** The framework proceeds to create and check out the new branch, then initializes your workspace.

Branch Creation

A new Git branch is automatically created based on your input and checked out, isolating your work from the default branch.

Workspace Initialization

The `.rdd-docs/workspace/` directory is prepared, and a fresh `work-iteration-prompts.md` file is generated from a template, ready for your first prompts.



Best Practice: Always start with a new, clean branch for each iteration. This prevents accidental changes to the main codebase and facilitates clear, atomic pull requests later in the workflow. It also ensures that Copilot's context is fresh and focused on the current task.

Working with Work Iteration Prompts

The core of RDD development revolves around the `work-iteration-prompts.md` file. This critical file, located at `.rdd-docs/work-iteration-prompts.md`, is where you define and manage the specific tasks that GitHub Copilot will execute during an iteration. It structures your development work into discrete, trackable, and executable units, allowing for a systematic approach to code generation and refinement. By carefully crafting these prompts, you guide Copilot through the entire development process, from initial feature implementation to testing and documentation.

Prompt File Structure & Components

```
# Work Iteration Prompts

## Prompt Definitions

- [ ] [P01] Create user authentication
module in `src/auth.py` with login
and logout functions. Include input
validation.

- [ ] [P02] Add unit tests for
authentication in `tests/test_auth.py`.
Cover valid login, invalid password,
and missing username cases.

- [x] [P03] Update README.md with
authentication usage examples
```

The `work-iteration-prompts.md` file is a Markdown document designed for clarity and ease of use. It typically begins with a main heading, followed by a "Prompt Definitions" section that lists all the individual prompts for the current iteration. Each prompt is a distinct line item with a clear, actionable directive for Copilot.

Each prompt entry includes the following essential components:

- **Checkbox:** Indicates the status of the prompt. `- [ ]` signifies a pending, unexecuted prompt, while `- [x]` marks a prompt that has been successfully completed by the RDD framework. This checkbox is managed automatically.
- **Prompt ID:** A unique identifier, such as `[P01]`, `[P02]`, etc. These IDs are crucial for referencing specific prompts and for the RDD framework to track execution progress.
- **Instructions:** A clear, concise, and detailed description of the task Copilot needs to perform. This should include specific file paths, expected functionalities, and any relevant constraints or guidelines.

> **Important:** The checkboxes (`[ ]` or `[x]`) should never be manually edited within the `work-iteration-prompts.md` file. The RDD framework is responsible for automatically updating a prompt's status to `[x]` once it has been successfully executed and verified by Copilot.

Writing Effective Prompts

Crafting effective prompts is key to leveraging RDD and GitHub Copilot efficiently. Good prompts are specific, actionable, and provide all necessary context for Copilot to generate accurate and relevant code. Here are examples of good versus bad prompts:

✓ Good Prompt Example

`[P01] Create a validation utility in `src/utils/validators.py` with functions for email validation (RFC 5322 format) and phone number validation (E.164 format). Include docstrings following Google style guide. Add comprehensive unit tests in `tests/utils/test_validators.py` with 100% coverage.`

This prompt is highly specific, providing exact file paths, detailing the required functionality (email and phone validation), specifying formats (RFC 5322, E.164), dictating code style (Google docstrings), and demanding comprehensive testing (unit tests, 100% coverage). It represents a single, well-defined task.

☒ Bad Prompt Example

`[P01] Add validation`

Why it's bad: This prompt is overly vague and lacks essential details. It doesn't specify what needs validation, where the code should be placed, what type of validation is required, or any quality standards. Such a prompt would likely lead to ambiguous or incomplete results from Copilot, requiring significant manual intervention or further prompting.

Always strive for clarity, specificity, and completeness when writing your prompts to maximize Copilot's effectiveness and minimize rework.

Executing Prompts in Copilot Chat

Once your prompts are defined in `work-iteration-prompts.md`, you can execute them using the Copilot Chat interface within VS Code. Follow these steps for execution:

1. **Open VS Code:** Ensure you have VS Code running with the GitHub Copilot extension installed and active.
2. **Launch Copilot Chat:** Open the Copilot Chat panel by pressing `Ctrl+Shift+I` (Windows/Linux) or `Cmd+Shift+I` (macOS).
3. **Select Appropriate Model:** Verify that you are using a capable large language model (LLM) such as Claude Sonnet 4 or GPT-4. The quality of Copilot's output heavily depends on the underlying LLM.
4. **Execute Prompts:** You have two primary methods for executing prompts:
   - **Execute a Specific Prompt:** To run a particular prompt by its ID, use the command: `/rdd.execute P01` (replace `P01` with the desired Prompt ID).
   - **Execute the Next Uncompleted Prompt:** To automatically run the next prompt that hasn't been marked as completed (i.e., its checkbox is `[ ]`), simply use the command: `/rdd.execute`. The framework will find the first uncompleted prompt and send its instructions to Copilot.
5. **Review and Iterate:** After Copilot executes a prompt, carefully review the generated code and make any necessary adjustments. This iterative process of prompting, executing, and refining is central to RDD.

Special Commands

The RDD framework includes two special commands designed for specific phases of your development workflow: documentation synchronization and comprehensive requirement analysis and planning.

The .rdd.update Command

The `.rdd.update` command is critical for maintaining accurate and current documentation. After completing development work on an iteration, this command scans your code changes and automatically updates the `requirements.md` and `tech-spec.md` files to reflect those changes. This ensures that your project documentation always stays in sync with your implementation, providing a reliable source of truth for your project.

Location: `.github/prompts/rdd.update.prompt.md`

When to Use: After all development prompts are executed for an iteration, and before marking the iteration as complete.

Usage: Simply execute `/rdd.update` in the Copilot Chat interface, or include it as the final prompt in your `work-iteration-prompts.md` file to run automatically.

What It Does:

- Scans all implementation files in your workspace for changes.
- Identifies new features, bug fixes, and documentation updates.
- Automatically updates `requirements.md` and `tech-spec.md` to reflect these changes.
- Helps maintain a synchronized and accurate project documentation.

The .rdd.analyze Command (Experimental)

The `.rdd.analyze` command is an experimental, interactive feature designed to help clarify complex requirements and generate a detailed execution plan. It guides you through a structured conversation to break down user stories into actionable prompts.

Location: `.github/prompts/rdd.analyze.prompt.md`

When to Use:

- At the beginning of a complex iteration where requirements are ambiguous.
- To generate a comprehensive execution plan with sequenced prompts.

> ⚠️ **WARNING:** This command is iterative and can consume a significant number of premium requests. It is highly recommended to first attempt manual clarification of the user story or use other tools before resorting to `.rdd.analyze`.

How It Works:

The command operates through a state-based workflow, guiding you through a series of steps to refine the user story and plan its implementation:

1. **State 1-4 (Clarification):** Systematically collects critical information by asking "What," "Why," "Acceptance Criteria," and "Other Considerations" related to your user story.
2. **State 5 (Questionnaire Generation):** Based on your input, it generates a comprehensive Requirements Questionnaire with multiple-choice questions to address any remaining ambiguities.
3. **State 6-7 (Response & Confirmation):** You answer the questionnaire, and the system confirms all clarity criteria are met.
4. **State 8-9 (Plan Generation):** A detailed execution plan is created, outlining sequenced prompts for your `work-iteration-prompts.md` file, ensuring a clear path to implementation.

> **Tip:** Always run `.rdd.update` before completing an iteration to keep your project documentation accurate and synchronized with implementation changes.

Completing Your Iteration

When all prompts are executed and documentation is updated, complete the iteration using menu option 3: **Complete current iteration**. You must be on a feature branch (NOT on default branch) and the workspace must NOT be empty.

Workspace Archiving

All files from `.rdd-docs/workspace/` are copied to `.rdd-docs/archive/<branch-name>/`. Archive metadata file created with timestamp, branch name, author, and commit info.

Push to Remote

If not in local-only mode, you're prompted to push branch to origin. Displays reminder to create pull request on GitHub.

1

2

3

4

Automatic Commit

All changes committed with message: "Completing work on <branch-name>". Includes code changes, documentation updates, and workspace files.

Return to Default

Checks out the default branch and clears workspace, making it ready for the next iteration.

What Happens Next

1. Create Pull Request

Use GitHub web interface to create a PR for your completed work. This happens outside the RDD framework.

2. Request Code Review

Assign team members to review your changes. Wait for approval before proceeding.

3. Merge & Cleanup

After PR approval and merge, use menu option 4 to delete the merged branch locally and keep your repository clean.