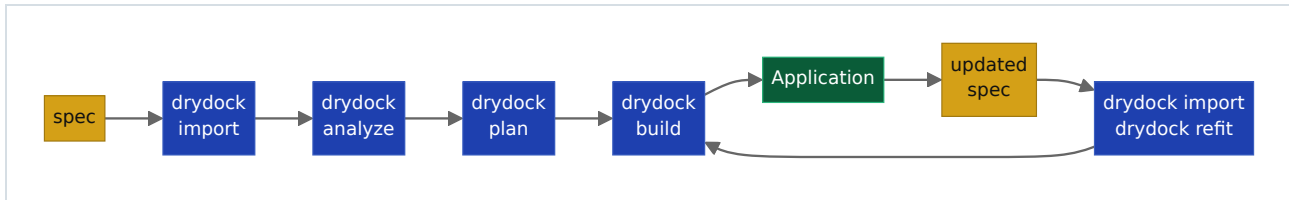


Quick Start Guide

Ed Barlow www.webcloudstudio.com August 2026



This quick start guide builds a trivial application called ReadingList.



- `import` copies your specification material into a workspace.
- `analyze` decomposes the import/epic into stories.
- `plan` grooms blueprints, ac, and the build graph.
- `build` creates software testing each step.
- `refit` diffs updated specs into tickets.
- The `quarterdeck` web interface provides control and observability.

Before you begin

1. Set up the `claude` or `codex` in your `PATH` (subscription API).
2. Install `uv` (easiest and its so fast.)

```
curl -LsSf https://astral.sh/uv/install.sh | sh          # macOS and Linux:
brew install uv                                           # macOS with Homebrew
winget install --id=astral-sh.uv -e                     # Windows with WinGet

# Windows PowerShell:
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

3. Install Drydock ([User Installation Guide](#)).

```
# Recommended – isolated tool install:
uv tool install drydock-sdd

# Alternative with `pipx`:
pipx install drydock-sdd

# Or if you want to manually setup your path:
python -m pip install drydock-sdd
```

`uv tool` and `pipx` install Drydock in a dedicated environment and put a command wrapper on your `PATH`; they are the right choice for an interactive CLI. `pip` installs

Drydock into whatever virtual environment is active and requires you to update your `PATH`.

4. Review your setup with

```
drydock --version
drydock config show           # show your configuration
    # drydock config set drydock_workspace <Workspace for the drydock>
    # drydock config set drydock_build_directory <Your application parent directory>
    # drydock is scoped to work only in these directories.
```

1. Create Target Workspace

Create workspace in `drydock_workspace/targets/<target>` and build target in `drydock_build_directory/<target>`.

```
# project ReadingList workspace in `drydock_workspace/targets/<target>`
# project ReadingList builds to `<drydock_build_directory>/ReadingList`
drydock init ReadingList           # Initialize project workspace
```

2. Import Your Specification

Create source specification material in a file (example: `reading-list.md`)

```
# Reading List

Build a web application that keeps a list of books to read.

The reader can add a book with a title and author, view the books in the order added,
and remove a book. An empty title or author is rejected with a clear error message.

The application includes automated tests for each behavior.
```

Import the file you just created:

```
drydock import ReadingList ./reading-list.md
drydock score spec ReadingList           # Optional
```

3. Analyze

The analyze command performs an initial analysis of the source material. Analyze turns the source material into stories, acceptance criteria, questions, and blockers.

```
drydock analyze ReadingList
drydock run quarterdeck ReadingList
```

Navigate to the QuarterDeck web server at the default address, <http://127.0.0.1:8080>. Review the Target, then press `Ctrl+C` in the terminal to stop the server and continue.

QuarterDeck provides the following screens:

- Commander's Chair - Overview of status
- Compass - Your constitution
- Analysis - The stories and input analysis
- Sea Trials - Project acceptance criteria
- Blockers - **If this exists, you must answer**
- Questionnaires - Discovery identity and stack choices

The Commander's Chair summarizes the analyzed stories, questionnaires, and blockers.

The Analysis page shows the stories and high-level acceptance criteria created from the source material.

SETUP
TECHNOLOGY_STACK.md

QUARTERDECK
ReadingList

Commanders Chair

Compass

Analyze Compass

Exclude Files

READINGLIST ANALYSIS

Analysis

Plan Compass

Technology Stack

Sea Trials

Discovery Identity

READINGLIST BUILD

Evidence

Blueprints

READINGLIST REFIT

Technology Stack

Approved ✓

The technology decisions of record for this target. One row per technology, naming the Rigging best-practice file that governs building it. Leave the Rigging cell as none when no Rigging file exists; the builder then applies general best practice. Analyze proposes this file once and never overwrites it; plan reads it to assign per-story stack guidance.

Technology Stack

Rows save automatically when you leave a field. Set Rigging to — when no Rigging guidance exists — the builder then applies general best practice.

Technology	Rigging	Notes
Python	python.md	Proposed conventional implementation languag
Flask	flask.md	Proposed lightweight web framework for the d
Bootstrap 5	bootstrap5.md	Proposed presentation framework for a simple
SQLite	sqliite.md	Proposed local persistence store for the readi
pytest	—	Proposed automated test runner for the requir
uv and Ruff	uv_ruff.md	Proposed environment and linting workflow; n

+ Add technology

4. Plan

The plan or Agile Decomposition stage creates **Blueprint** files which are buildable specifications and the **Manifest** or dependency-aware build plan. The Blueprints contain test driven development assertions.

In QuarterDeck you have access to:

From QuarterDeck you can see the stages of the build and the details of each stage. Drydock breaks the build into Blocks of similar work: foundational work, persistence work, services, and service consumers such as UI screens.

SETUP
MANIFEST .md

4 Build Blocks 10 stories 0 built 3 ready to build 7 blocked 0 failed Blueprint SP 7,189 - Context SP 63,166 - Used SP 46,118

Buildable now: **block-1**

Optimize build blocks

BUILD BLOCK

Block 1 • Foundational

Combined Story Points = 15,901 Story Point Savings = 2,586 0/3 verified

READY TO BUILD
STORY
Foundation

Establish the Flask application structure and module boundaries.

Story Points = 8,749 (overhead 7,966)

Direction: Establish the Flask application factory, configuration boundary, route registration structure, and module ownership boundaries needed by the reading-list application. Keep persistence access behind the database interface and make the application runnable for downstream stories.

- ▶ definition of done — 3 programmatic checks
- ▶ plan and traceability
- ▶ stack breakdown

READY TO BUILD
STORY
Foundation

Establish SQLite persistence for ordered book records.

Story Points = 3,938 (full 5,998, overhead 5,075)

Establish SQLite persistence for ordered book records.

Direction: Implement the SQLite database boundary and BookRepository described by DATABASE.md. Provide creation, ordered retrieval, and deletion operations for books with an integer identifier, title, author, and insertion order represented by the identifier.

Depends on: Establish the Flask application structure and module boundaries. (pending)

- ▶ definition of done — 4 programmatic checks
- ▶ plan and traceability
- ▶ stack breakdown

READY TO BUILD
STORY
Foundation

Establish shared Bootstrap presentation patterns for the reading-list interface.

Story Points = 3,214 (full 3,740, overhead 3,148)

Establish shared Bootstrap presentation patterns for the reading-list interface.

Direction: Establish the shared Bootstrap 5 layout, typography, navigation, feedback-message styling, and responsive form/list patterns used by the reading-list screen.

Depends on: Establish the Flask application structure and module boundaries. (pending)

- ▶ definition of done — 3 programmatic checks
- ▶ user acceptance
- ▶ plan and traceability
- ▶ stack breakdown

BUILD BLOCK

Block 2 • Service

Combined Story Points = 9,094 Story Point Savings = 17,232 0/4 verified

BLOCKED
STORY
Feature/Service

Validate required title and author values before persistence.

Story Points = 4,285 (overhead 3,634)

Blocked by story Establish the Flask application structure and module boundaries.

READINGLIST
 ANALYSIS

Analysis

Plan Compass

Technology Stack

Sea Trials

☒ Discovery Identity

READINGLIST
 BUILD

MANIFEST

Kanban Board

Decisions

Evidence

Blueprints

READINGLIST
 REFIT

Refit

5. Build

The created application is in `<drydock_build_directory>/ReadingList/`.

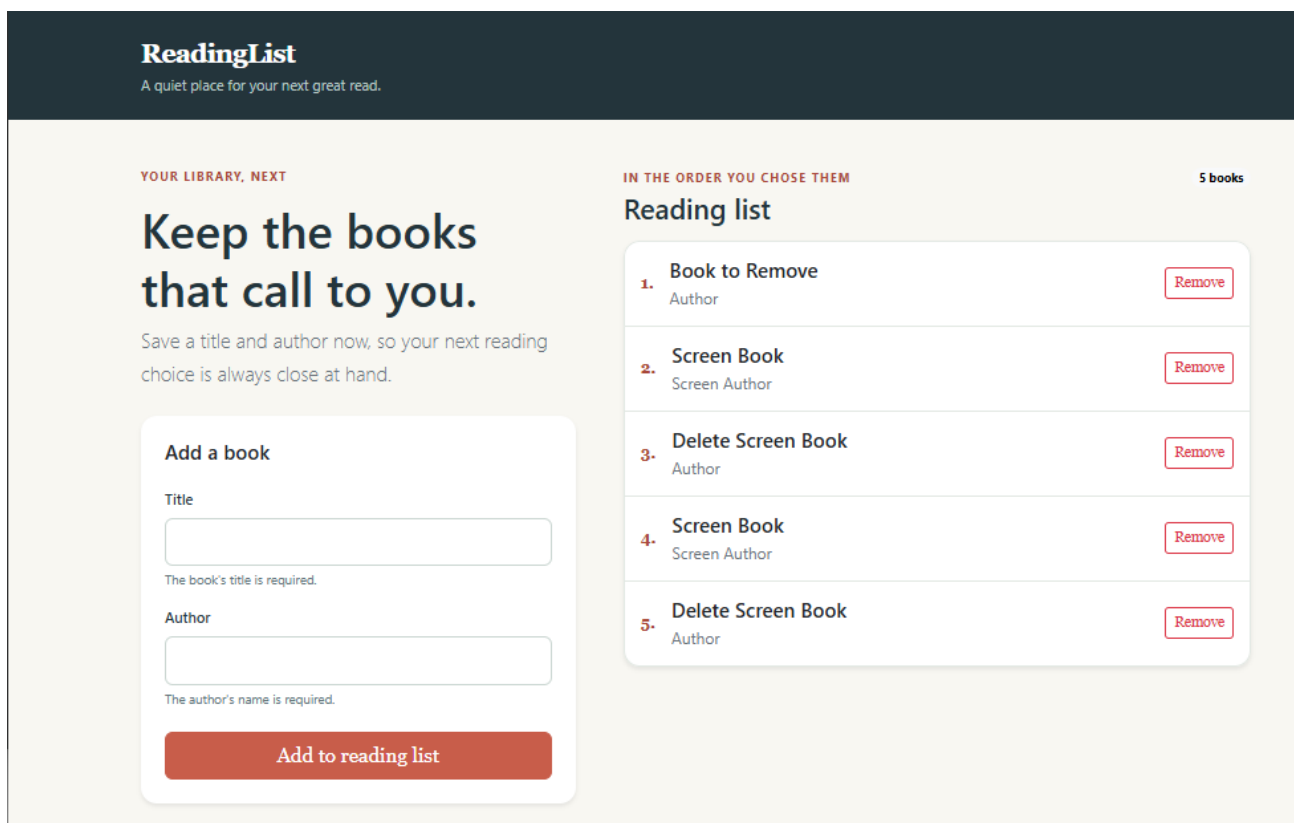
```
cd <drydock_build_directory>/ReadingList
uv run run.py
```

```

barlo@MSI:/mnt/c/Users/barlo/projects/ReadingList$ ls
README.md  __pycache__  app  bin  config.py  data  pyproject.toml  pytest.ini  run.py  tests  uv.lock
barlo@MSI:/mnt/c/Users/barlo/projects/ReadingList$ uv run run.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5001
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 518-695-593

```

Follow the generated project instructions to start the application; this example runs it with `uv run run.py`.



The generated ReadingList application running after the build.

6. Refit - Changing the Application

Add the following line to `reading-list.md`:

The reader can mark a book as read and view whether each book is unread or read.

Re-import the source, generate Refit Tickets, and run the incremental build:

```

drydock import ReadingList --update # Re-import your updated specs
drydock refit ReadingList --sources # Create Refit Tickets, Update the Manifest
drydock build ReadingList          # Incremental Build

```

This is Drydock's development workflow, designed for high-velocity changes to specifications. `drydock refit` uses Git diff to identify source-material changes, maps source changes to Blueprints, and appends work to the build graph, enabling `drydock build` to run normally.

The post-production workflow uses a similar process but treats the internal Blueprints, rather than the user specifications, as canonical. Production change tickets map directly to Blueprints to minimize drift and make replanning mechanical without an LLM.

BUILD BLOCK Block 5 • Feature Combined Story Points = 1,822 Story Point Savings = 996 0/3 verified

READY TO BUILD **STORY** Add the route and application behavior for marking a selected book as read. Story Points = 899 (overhead 590)
Add the route and application behavior for marking a selected book as read.
Depends on: Add persisted read state per book, update the book persistence interface, and migrate existing rows as unread. (pending)
► plan and traceability
► stack breakdown

READY TO BUILD **STORY** Add persisted read state per book, update the book persistence interface, and migrate existing rows as unread. Story Points = 435 (full 933, overhead 597)
Add persisted read state per book, update the book persistence interface, and migrate existing rows as unread.
► plan and traceability
► stack breakdown

READY TO BUILD **STORY** **Screen** Render each book's read or unread state and provide the mark-read control. Story Points = 488 (full 986, overhead 645)
Render each book's read or unread state and provide the mark-read control.
Depends on: Add the route and application behavior for marking a selected book as read. (pending)
► plan and traceability
► stack breakdown

The updated Manifest with new stories.

ReadingList
A quiet place for your next great read.

YOUR LIBRARY, NEXT

Keep the books that call to you.
Save a title and author now, so your next reading choice is always close at hand.

Add a book

Title

The book's title is required.

Author

The author's name is required.

Add to reading list

IN THE ORDER YOU CHOSE THEM **5 books**

Reading list

Book to Remove

1. Author Unread **Remove** **Mark as read**

Screen Book

2. Screen Author Unread **Remove** **Mark as read**

Delete Screen Book

3. Author Unread **Remove** **Mark as read**

Screen Book

4. Screen Author Unread **Remove** **Mark as read**

Delete Screen Book

5. Author Unread **Remove** **Mark as read**

The incrementally rebuilt application lets the reader mark books as read and see their status.

Links

- [Project Home Page](#)
- [GitHub](#)
- [User Installation Guide](#)
- [Drydock Specification](#)

Copyright © 2026 Web Cloud Studio.