Issue Management with Visual Builder Studio Issues

Oracle Fusion Implementation — Ways of Working

Audience: Project Manager, Functional Analysts, Customer Approvers, Developers / Technical Team

Purpose: A shared, practical guide to how we record, track, and move work through delivery and deployment using Visual Builder Studio (VBS) Issues.

Contents

Contents 2
1. Overview 3
2. Key Terms 4
3. The Release and Component Fields 5
3.1 Component — which track the work belongs to 5
3.2 Release — which environment the work targets 5
4. How We Structure Work 6
4.1 Worked example 6
5. Issue Types — How to Use Each 7
6. Linking Issues (Relations) 8
7. Tracking Deployment Across Environments 9
7.1 The approve-then-deploy cycle 9
7.2 Reading the current state at a glance 9
8. Sprints and the Backlog 11
8.1 How Tasks flow 11
8.2 What goes in a sprint — and what does not 11
8.3 Planning rhythm 11
9. Boards We Use 12
10. What Each Role Does 13
11. Enabling Email Notifications 14
11.1 Open your preferences 14
11.2 Verify your email address 14
11.3 Set your notification preferences 14
12. Conventions to Keep It Clean 15
13. Good to Know (Current Limits) 16

1. Overview

We use Visual Builder Studio (VBS) Issues as the single system of record for all technical delivery work on this implementation: extensions, reports, customizations, and integrations. Every piece of delivered functionality is tracked from design through development, QA, and deployment into each environment.

This document explains the structure so that everyone — regardless of role — reads the board the same way and records work consistently. Three ideas underpin everything that follows:

- **Hierarchy:** each delivered item (a RICE element) is an Epic, with its activities tracked as child Tasks beneath it.
- **Build lifecycle:** each activity moves through statuses we can see on a board and plan into sprints.
- **Deployment promotion:** a deliverable is approved into and then deployed to each environment, in sequence, with a clear record of where it currently is.

2. Key Terms

A shared vocabulary so the document reads the same for everyone.

Term	What it means here
RICE	Reports, Interfaces, Conversions, Extensions — the categories of technical deliverables on a Fusion implementation. Each individual deliverable is referred to as a RICE element or RICE item.
Epic	The container for one RICE deliverable. The unit the customer signs off on.
Task	A unit of work under an Epic — either a build activity (e.g. Development) or a deployment step.
Component	The functional/technical area an item belongs to (e.g. P2P, O2C, R2R, FA, INV, CM, Localization, Infrastructure). Used to slice and filter.
Rice#	The unique identifier for a deliverable, recorded on its Epic.
Environment	A target Fusion environment in the promotion path: P1, P2, UAT, SIT, PROD.
Board	A visual, column-based view of issues, used to see status at a glance.
Sprint	A time-boxed planning period that build Tasks are scheduled into.

3. The Release and Component Fields

Two standard fields classify every issue so the work can be filtered, reported, and routed. Set both on each Epic when it is created.

3.1 Component — which track the work belongs to

Component identifies the functional or technical track an item sits in. It is what lets us slice every board and report by area — for example, all open work in P2P, or all defects in O2C. The values are the delivery tracks plus one cross-cutting technical track:

Component	Use it for
P2P	Procure-to-Pay track — deliverables in the purchasing/payables process area.
O2C	Order-to-Cash track — deliverables in the order/receivables process area.
R2R	Record-to-Report track — deliverables in the general ledger / financial reporting area.
FA	Fixed Assets track.
INV	Inventory track.
CM	Cash Management track.
Localization	Country/statutory localization work that applies across process areas.
Infrastructure	Cross-track functional and technical infrastructure components.

Rule of thumb: work in a single functional process uses that track; work that underpins the platform as a whole uses Infrastructure.

3.2 Release — which environment the work targets

The Release field records the environment an issue is associated with, from the same ladder used for deployment. It complements the deployment fields on the Epic (Section 7): Release marks the environment context, while the deployment fields track approval and install progress.

Release	What it represents
P1	The first build/playback environment — earliest target where a deliverable is assembled and first demonstrated.
P2	The second build/playback environment — the next iteration target after P1.
UAT	User Acceptance Testing — where the customer validates the deliverable against requirements.
SIT	System Integration Testing — where end-to-end and cross-system behaviour is verified.
PROD	Production — the live environment.

Note: Release is a single value per issue and reflects the environment context, not a running history. The full promotion trail is captured by the deployment fields and audit log (Section 7).

4. How We Structure Work

Each delivered functionality is an Epic. Beneath it sit the activity Tasks that carry it from design to customer acceptance, plus the deployment Tasks that record promotion through environments.

Epic (one RICE deliverable — carries Rice#, Component)

- **Task:** Functional Design
- **Task:** Technical Design
- **Task:** Development
- **Task:** Functional QA
- **Task:** Customer QA
- **Defects** (raised during QA, linked to the Epic — see Section 5)

This keeps two things separate and clean: the Epic represents the deliverable as a whole, and the Tasks represent the individual activities that progress independently and can be planned into sprints.

4.1 Worked example

Here is how a single deliverable looks once it is set up. The Epic carries the identifying fields; the activity Tasks sit beneath it and move through their statuses independently.

Epic: RICE-014 — Print Receivables Transaction

Field	Value
Issue Type	Epic
Rice#	RICE-014
Component	O2C (Order-to-Cash) — a receivables transaction print belongs to the order/receivables track
Release	P1 (current target environment for first build and playback)
Deployment Environment	P1
Deployment Done?	Unchecked (approved for P1, not yet installed)
Summary	Print Receivables Transaction — formatted transaction print for the receivables process

Child Tasks under this Epic:

- **Task:** Functional Design — define the print layout, data fields, and triggering rules
- **Task:** Technical Design — BI Publisher template approach, data model, and integration points
- **Task:** Development — build and unit-test the report and template
- **Task:** Functional QA — validate output against the design
- **Task:** Customer QA — customer reviews and accepts the deliverable

As work progresses, each Task moves through its statuses on the Delivery board, and the Epic's deployment fields advance through each environment as the customer approves and the technical team installs (Section 7).

5. Issue Types — How to Use Each

VBS Issues offers several issue types. We use them with specific, agreed meanings so reporting stays consistent.

Issue Type	Use it for	Notes
Epic	One RICE deliverable — the parent container for all of its activities.	Carries Rice# and Component. The customer signs off at this level. Epics span sprints, so they are not added to sprints themselves.
Task	The five build activities and the deployment steps under an Epic.	This is the everyday work item. Tasks appear on boards and are planned into sprints.
Defect	A bug found during Functional QA, Customer QA, or after deployment.	Linked to its Epic (or the Development Task). Kept separate from Tasks so quality metrics stay clean.
Story	Optional — a unit of build work where we want agile estimation/sizing.	Use only if the team chooses to size work. Otherwise the five Tasks are sufficient.
Feature	Optional — a grouping when one large RICE deliverable splits into independently shippable sub-parts.	Most deliverables go Epic → Task directly. Do not force a Feature layer where it adds no value.

6. Linking Issues (Relations)

VBS lets us link issues to one another. To keep the picture readable, we standardise on a small set of relation types, each with one agreed meaning:

Relation	Use it for
Implements / Implemented By	Tying activity Tasks to their parent Epic — the Tasks implement the deliverable.
Depends On / Dependency Of	Build sequencing. Within an Epic (Development depends on Technical Design) and across Epics (one deliverable cannot be built until another exists).
Requires / Required By	A hard prerequisite — a true blocker where one item genuinely cannot function without the other. Use sparingly, reserved for real blockers.
Discovered While Testing	Linking a Defect back to the deliverable whose testing surfaced it — gives defect provenance.
Relates To	General, informational links only (e.g. two items touching the same Fusion object). Use sparingly so it stays meaningful.

Note: relations are recorded and navigable, but VBS does not automatically block work when a dependency is unmet. They inform planning; they do not enforce it.

7. Tracking Deployment Across Environments

Deployment moves a deliverable up a defined ladder of environments, with the customer approving entry into each environment and the technical team performing the actual install. We track this on the Epic using three fields:

Field	Meaning
Deployment Environment	Single-select list (P1, P2, UAT, SIT, PROD). The environment the deliverable is currently being promoted into.
Deployment Done?	Checkbox. Unchecked = approved for the environment, not yet installed. Checked = installed in that environment.
Rice#	The deliverable's identifier, for filtering and cross-reference.

7.1 The approve-then-deploy cycle

For each environment, two distinct events happen, performed by two different roles:

1. **Customer approves** entry into the environment → set Deployment Environment to that environment, leave Deployment Done? unchecked.
2. **Technical team installs** the deliverable → check Deployment Done?.
3. **Customer approves the next** environment → advance Deployment Environment, uncheck Deployment Done? again.

Repeating this up the ladder (P1 → P2 → UAT → SIT → PROD) walks the deliverable through every gate. The fields always show where the deliverable is right now; the full history of these changes is preserved automatically in the issue's audit trail.

> **Important — clear the checkbox when you change environment.** Whenever Deployment Environment is advanced to the next environment, Deployment Done? **must be unchecked** at the same time. The checkbox always describes the environment currently shown in Deployment Environment. If it is left checked after the environment changes, the item will read as “already installed” in the new environment when it is not — which silently drops it from the technical team’s deployment queue. Advancing the environment and clearing the checkbox is a single action, never two separate ones.

7.2 Reading the current state at a glance

Deployment Environment	Deployment Done?	What it means / whose turn
P1	Unchecked	Approved for P1, not yet installed — technical team's queue.
P1	Checked	Installed in P1 — awaiting customer approval for the next environment.
UAT	Unchecked	Approved for UAT, not yet installed — technical team's queue.

Deployment Environment	Deployment Done?	What it means / whose turn
PROD	Checked	Live in production.

Because these are standard single-value fields, we can search and build boards directly on them — for example, “everything approved for P1 but not yet deployed.”

8. Sprints and the Backlog

Build work is planned and tracked through sprints. The backlog is the holding area for everything not yet scheduled.

8.1 How Tasks flow

Every Task starts life in the backlog. When we plan a sprint, we move the Tasks we intend to work on in that period into the sprint. A Task can therefore be in one of two states at any time:

- **In the backlog** — identified but not yet scheduled. The backlog is the full, prioritised list of pending work across all Epics.
- **Associated with a sprint** — committed to a specific time-boxed period, where the team actively works it to completion.

Each Task is associated with at most one sprint at a time. Moving a Task into a sprint is how we commit to delivering it in that period; leaving it in the backlog keeps it visible and prioritised without committing it yet. Tasks not finished by the end of a sprint return to the backlog (or roll to the next sprint) for re-planning.

8.2 What goes in a sprint — and what does not

Item	Sprint behaviour
Activity Tasks (Functional Design → Customer QA)	Planned into sprints — this is the everyday sprint content.
Deployment steps	Can be scheduled into a sprint when an install is planned for that period, or tracked on the deployment board independently.
Defects	Planned into sprints alongside Tasks when they are picked up for fixing.
Epics	Not added to sprints — they span multiple sprints. Plan at the Task level; the Epic aggregates the progress of its Tasks.

8.3 Planning rhythm

In practice the cycle is: prioritise the backlog → pull the top items into the next sprint → work them through their statuses on the board → close the sprint, returning anything unfinished to the backlog. The backlog is reviewed regularly so priorities stay current as the implementation evolves.

9. Boards We Use

Boards give each role the view they need. The same underlying issues feed all of them. A board typically shows the Tasks of the current sprint arranged by status, so progress is visible at a glance as cards move left to right.

Board	Columns	Who relies on it
Delivery / Build	The five activities by status (Functional Design → Customer QA).	Analysts and developers — shows where each deliverable is in the build lifecycle.
Deployment / Promotion	Grouped by Deployment Environment; split by Deployment Done?	Technical team (what to install) and customer (what to approve next).
Defects	Defects by status.	Everyone — open quality items, by environment or component.

Swimlanes by Component (P2P, O2C, R2R, etc.) or by Rice# let any board be sliced by track. A board can be scoped to the active sprint (what we are working now) or to the wider backlog (everything pending).

10. What Each Role Does

Role	Responsibilities in the system
Project Manager	Monitors progress across Epics and boards; tracks deployment readiness and defect trends; uses Component/Rice# slices for status reporting.
Functional Analyst	Authors Functional Design Tasks; performs Functional QA; raises Defects with clear reproduction detail and links them to the deliverable.
Customer Approver	Performs Customer QA; approves entry into each environment by advancing Deployment Environment; signs off the deliverable at the Epic level.
Developer / Technical Team	Authors Technical Design; performs Development; installs into each environment and checks Deployment Done?; resolves Defects.

11. Enabling Email Notifications

So that everyone stays informed of changes to the issues they care about, each user must enable email notifications in their own Visual Builder Studio profile. This is a one-time setup each person does for themselves. There are two parts: first verify your email address, then choose which events you want to be notified about.

11.1 Open your preferences

Sign in to Visual Builder Studio, open the account menu at the top-right (your initials), and choose Preferences.

11.2 Verify your email address

On the Profile tab, check the Email Address field. It must show your correct address and be marked Verified. If it is not verified, follow the verification prompt (you will receive an email with a confirmation link) before continuing — notifications are only sent to a verified address.

11.3 Set your notification preferences

Open the Notifications tab and select the events you want to be notified about. At minimum we recommend enabling:

- **Issues updates, attachments, and comments** — so you are alerted when an issue you are involved in changes.
- **Project updates** — for project-level changes that affect everyone.
- **Include my own updates** — optional, if you want a record of your own changes too.

Other categories (build activities, SCM/Push, wiki) can be left off unless they are relevant to your role. Your choices are saved automatically.

Once verified and configured, you will receive email for the events you selected — keeping the whole team aware of issue activity without having to poll the boards.

12. Conventions to Keep It Clean

A few simple disciplines keep the system reliable as the number of issues grows:

- **Always set Component and Rice# on every Epic** from the moment it is created — this is what makes boards and reports sliceable later.
- **Advance deployment one environment at a time** and follow the approve-then-deploy order. The system does not enforce the sequence; we do.
- **Use the agreed relation types** with their agreed meanings (Section 5) rather than improvising.
- **Keep Defects as Defects,** not Tasks, so quality reporting stays accurate.
- **When advancing Deployment Environment, clear Deployment Done?** in the same action (see Section 7.1) — never leave it checked against the new environment.
- **Keep the backlog prioritised** and only commit Tasks into a sprint when the team intends to work them that period.
- **Write descriptive summaries** — a reader should understand an issue from its title and the Epic it sits under.

13. Good to Know (Current Limits)

Setting expectations honestly so no one is surprised:

- Epics are not added to sprints — they span sprints. Plan and burn down at the Task level; the Epic aggregates.
- Boards display child issues (Tasks, Defects), not Epics. This is why deployment is tracked on fields the Tasks/Epic carry and surfaced through search and board columns.
- Deployment fields show the current state. The change history lives in each issue's audit trail — readable per issue, but not designed for bulk historical reporting across many issues. If point-in-time history across the whole project becomes a need, we will add per-environment deployment Tasks at that point.
- The system records dependencies but does not block work when one is unmet — relations guide planning, they do not enforce it.

This is a living document. As the project matures and reporting needs evolve, the approach will be revisited and this guide updated.