

Dependency Radar for Jira

Client Installation & Admin Guide and User Guide

Dependency Radar is a read-only Jira Cloud app that turns the issue links your squads already maintain into Sprint and Matrix dependency risk views. It is designed for multi-squad Agile delivery groups and their scrum-of-scrums conversations, helping scrum masters, delivery managers, and squad leads spot blocked, at-risk, and carried-over cross-squad work without leaving their normal Jira workflow.

Contents

- [Part 1 — Client Installation & Admin Guide](#)
- [Part 2 — User Guide](#)
- [Support and contact](#)

Part 1 — Client Installation & Admin Guide

For Jira administrators and delivery leaders.

1. Before you start

- A Jira Cloud site with site administrator or app installation permission.
- Projects and boards already in use by the squads you want to visualise.
- Issue links maintained between work items (for example, depends on / is depended on by).
- Due dates (or start dates) populated on linked work items so sprint placement can be derived.

2. Installation

- Open the Atlassian Marketplace listing for Dependency Radar, or search for “Dependency Radar” from Jira > Apps > Explore more apps.
- Select Get app / Try it free and confirm the installation into your Jira Cloud site.
- Review the requested permissions on the consent screen and approve.
- After installation the app appears under Apps in the Jira top navigation.

3. Requested permissions

Dependency Radar is a Forge app and requests the minimum scopes needed to render dependency views:

- Read Jira work item data — issue keys, summaries, statuses, projects, dates, and issue links.
- Read project and board data — to group work by squad and align it to sprints.
- App storage — to save your view configuration (squads, thresholds, sprint calendars).

The app is read-only against Jira: it never creates, edits, transitions, or deletes work items, and it does not copy Jira content to external systems.

4. First-time configuration

Open Apps > Dependency Radar and work through the configuration panel.

4.1 Sprint calendar

- Set the sprint length and the start date of your first sprint so columns align to your cadence.
- Name sprints to match the naming your delivery group already uses.

4.2 Matrix periods

- Choose the period the matrix view aggregates over (for example, the current and next two sprints).
- Adjust the period whenever your planning horizon changes.

4.3 Link mappings

- Map your Jira link types to provider and consumer relationships (for example, depends on = consumer, is depended on by = provider).
- Exclude link types that are not delivery dependencies, such as relates to or duplicates.

4.4 Date fields

- Select which date field drives sprint placement — typically Due date, optionally Start date.
- Ensure linked work items carry that date, otherwise they cannot be placed on the sprint timeline.

4.5 Squads and thresholds

- Define squads from projects, boards, or components so rows in the views match your delivery structure.
- Set the risk thresholds that drive amber and red flags (for example, days of slippage before an item is flagged).
- Save the configuration as a named view for a squad group or ART-sized delivery group.

5. Validation checklist

- The app loads under Apps > Dependency Radar for a standard (non-admin) user.
- Every squad you expect appears as a row in the Sprint view.
- A known cross-squad dependency appears with the correct provider and consumer.
- Sprint columns align to your real sprint dates.
- A deliberately overdue linked item shows as red or amber.
- Items without a due date are visible in the unplaced/undated grouping rather than silently dropped.
- Matrix view totals reconcile with the Sprint view for the same period.

6. Administration and troubleshooting

- A squad is missing — check the squad definition and that the user has Jira browse permission on those projects.
- A dependency is missing — confirm the link type is mapped and both work items are in scope of the view.
- An item sits in the wrong sprint — check the configured date field on the item and the sprint calendar start date.
- Different users see different data — this is expected: Dependency Radar respects each user's Jira permissions.
- Removing the app — uninstalling from Manage apps removes stored configuration; Jira data is unaffected.

Part 2 — Dependency Radar User Guide

Working app name: Dependency Radar. Audience: scrum masters, scrum-of-scrums facilitators, delivery managers, release leads, product owners, squad leads, and other users who need a clearer view of cross-squad delivery dependencies in Jira.

What Dependency Radar is for

Dependency Radar is a read-only Jira Cloud app that turns existing Jira issue links into delivery dependency views. It helps teams see which squads are waiting on each other, where timing risk is building, and which linked issues need attention before they affect sprint or release outcomes.

The app does not create dependencies. It reads Jira issue links, dates, statuses, projects, and configured delivery rules, then presents them in two planning views:

- Sprint view: dependency risk by squad across the current and upcoming sprint buckets.
- Matrix view: squad-to-squad dependency concentration during the configured matrix period.

Jira remains the source of truth. Dependency Radar is a planning and conversation layer over Jira, not a replacement for Jira issue linking, Advanced Roadmaps / Plans, or team delivery discipline.

Getting started

Use this short path the first time you open Dependency Radar:

- 1 Open Dependency Radar from Jira.
- 2 Choose an existing saved view if one already matches your squad group.
- 3 If there is no suitable saved view, open Edit view.
- 4 Select the Jira projects that represent the squads you want to monitor.
- 5 Select the dependency link types the group uses.
- 6 Choose the direction filter for Sprint view: Incoming, Outgoing, or Both.
- 7 Save the view. Use a private view for personal investigation, or a shared view for a recurring delivery group.
- 8 Check Sprint view first to confirm dependencies appear in the expected squad rows and sprint buckets.
- 9 Switch to Matrix view to check the selected squads' provider-consumer relationships.
- 10 Open red, amber, or grey items and follow the Jira links when issue data needs correction.

If the board looks empty, start by checking the saved view scope. Dependency Radar needs selected projects, configured link types, and existing Jira issue links before it can show useful dependency risk.

Why users typically open the app

Users usually come to Dependency Radar because they need to answer one of these questions quickly:

- Which squads are blocked or at risk because they are waiting on another squad?
- Which squad is creating the most downstream pressure for others?
- Which dependencies need discussion in scrum-of-scrums today?
- Are there red or amber dependencies in the current or next few sprints?
- Which dependencies have missing dates, unclear direction, or weak Jira hygiene?
- Are our Jira issue links telling the same story that teams are telling in planning conversations?

- What needs follow-up before a sprint, release, or quarterly planning checkpoint?
- Are old open dependencies still relevant, or are they stale Jira noise?
- Do we have enough dependency information to trust the plan?

The app is most useful when the user is not trying to manage every issue in detail, but trying to focus a coordination conversation on the dependency relationships that matter.

The user mental model

Dependency Radar works best when users think in four layers:

- 1 Jira issues are still the work records.
- 2 Jira issue links describe dependency relationships.
- 3 Global settings tell the app how to interpret links, dates, statuses, risk, sprints, and matrix periods.
- 4 Saved views tell the app which squads and link types a particular planning group cares about.

If a dependency is missing, unexpected, grey, or shown in the wrong place, the cause is usually in one of those layers: the Jira issue data, the Jira link, the global settings, or the saved view.

Who uses which part

- Most users can use saved views, Sprint view, Matrix view, risk counters, dependency details, and Jira issue links without changing global settings.
- Jira administrators or app administrators maintain the shared interpretation rules: dependency link mappings, date fields, sprint calendar, risk thresholds, Done statuses, Blocked statuses, stale-open rules, and matrix period.
- Saved view owners maintain the scope for a particular group: selected squads, dependency direction, included link types, view name, description, and whether the view is private or shared.

Setting up the first saved view

A saved view is the user's working lens over Jira dependency data. It tells the app which squads, link types, and Sprint view direction should be shown together.

To set up the first useful saved view:

- 1 Open Edit view.
- 2 Give the view a clear name, such as the scrum-of-scrums group, release train, programme, or delivery area.
- 3 Add a short description so other users know when to use it.
- 4 Select the Jira projects that represent the squads in the group.
- 5 Select the link types that represent delivery dependencies for that group.
- 6 Choose Both for the first pass unless the group deliberately wants only incoming or outgoing dependencies in Sprint view.
- 7 Save the view.

Use Save as new when experimenting with a different squad set, rather than overwriting a shared operational view.

After saving, check at least one known dependency. Confirm that it appears in the expected Sprint view row and, if both squads are selected, the expected Matrix view cell.

For a recurring ceremony, the first shared view should usually include all squads that regularly attend that ceremony. If the group often discusses dependencies with squads outside the ceremony, consider whether those provider or consumer squads should also be included, otherwise those relationships may not appear in Matrix view.

Main workflow

Use this workflow when preparing for or running a delivery coordination meeting:

- 1 Open Dependency Radar from Jira.
- 2 Choose the saved view for the relevant squad group.
- 3 Check the summary counters for red, amber, grey, and green dependencies.
- 4 Start with Sprint view if the discussion is about timing across current and upcoming sprints.
- 5 Switch to Matrix view if the discussion is about squad-to-squad pressure or ownership relationships.
- 6 Use the RAG counters to focus the view on At risk, Watch, Unknown, or Healthy dependencies.
- 7 Open dependency cards or matrix cells to inspect the consumer issue, provider issue, linked projects, direction, inferred date, and risk reasons.
- 8 Open the underlying Jira issues when ownership, date, status, or link data needs correction.
- 9 Record follow-up actions in Jira or in the team's normal delivery process.
- 10 Refresh the app after Jira changes if the group needs to confirm the updated dependency view.

The app should make the right conversation easier. It is not the place to rewrite the plan or maintain duplicate dependency records.

Before a scrum-of-scrums session

Before a live session, a facilitator should:

- Load the saved view that matches the group attending the session.
- Confirm the selected squads look right.
- Scan the red and amber counts.
- Scan the grey count to see whether missing data may weaken the conversation.
- Check whether stale-open dependencies are visible, hidden, or isolated, depending on the configured display mode.
- Open Matrix view to see whether one squad-to-squad relationship dominates the risk.
- Prepare two or three dependency relationships for discussion instead of trying to cover every link.

Good use of the app is selective. The goal is to identify the dependency relationships that need human coordination.

During a scrum-of-scrums session

During the session, use Dependency Radar to drive short, evidence-based questions:

- Who is the consumer squad waiting for?
- Who is the provider squad?
- What issue represents the provider work?
- What issue represents the consumer work?

- Is the provider expected before the consumer needs it?
- Is the risk caused by timing, status, missing data, or stale Jira hygiene?
- Who owns the next action in Jira?
- Does the saved view need another squad added to make the dependency visible?

Avoid treating the app as a verdict. It shows risk based on configured Jira data. Users still need to apply delivery judgement.

Sprint view

Sprint view shows dependency risk across the current and upcoming sprint buckets. It is designed for timing conversations.

Use Sprint view when you want to know:

- Which dependencies matter in the current sprint window?
- Which squad rows have red or amber dependency pressure?
- Which dependency cards need follow-up before the next sprint starts?
- Which dependencies are missing enough date data to be trusted?
- Whether a provider-done dependency is still relevant because the consumer work is active in the visible sprint window.

Rows represent selected squad projects. Sprint columns are generated from the configured sprint calendar. Dependency cards appear where the app can place the displayed row issue into the visible sprint window.

Sprint view intentionally does not force every dependency into the first visible sprint. If the app cannot find a usable placement date, or if the displayed row issue only has dates outside the visible window, the dependency can be omitted from the normal Sprint view.

Matrix view

Matrix view shows dependency concentration between selected squads. It is designed for scrum-of-scrums relationship conversations, not long-form dependency history.

Use Matrix view when you want to know:

- Which squads are depending on which other squads?
- Which provider squads are creating the most downstream pressure?
- Which consumer squads are most exposed to external dependencies?
- Which squad-to-squad relationships carry red, amber, or grey risk?
- Whether the group's selected squads are enough to explain the current dependency picture.

Matrix layout:

- Rows show consumer squads.
- Columns show provider squads.
- A non-empty cell means the row squad depends on the column squad.
- Cell colour follows the highest-risk dependency in that relationship.
- Clicking a non-empty cell opens a read-only dependency list.
- The Rows / columns control can show all selected squads or hide empty rows and columns.

- Unowned or unmapped dependencies appear separately from the main matrix when present.

Matrix view uses the same saved squad selection and link-type configuration as Sprint view. It uses a separate matrix period configured by the app administrator.

Matrix view only shows dependencies between squads selected in the current saved view. If a dependency exists to a squad that is not selected for display, that dependency will not appear in the main matrix.

This is an important difference from Sprint view. Sprint view starts from the selected squad rows and can still show incoming or outgoing dependencies for those squads even when the linked squad is outside the saved view. Matrix view is stricter because both the consumer squad and provider squad need to be in the selected project set before their relationship can appear as a cell.

Choosing between Sprint view and Matrix view

Use Sprint view when the question is about timing:

- What is at risk in the current or upcoming sprint?
- Which sprint bucket contains the dependency conversation?
- Is there enough slack between provider readiness and consumer need?

Use Matrix view when the question is about relationships:

- Which squads need to talk?
- Which provider squad is creating pressure?
- Which consumer squad is exposed?
- Which squad-to-squad relationship has the highest risk concentration?

In practice, facilitators often start with Matrix view to identify the key relationship, then move to Sprint view to inspect timing.

Switching between views

Use the View mode control to switch between Sprint and Matrix. Switching views does not change Jira data or the saved view. It only changes the visual lens over the selected projects and link types.

When you switch:

- Sprint view shows dependency cards by squad row and sprint bucket.
- Matrix view shows squad-to-squad relationship cells for the configured matrix period.
- The RAG counters continue to help focus on at-risk, watch, unknown, or healthy dependencies.
- The direction quick filter applies to Sprint view. Matrix direction is shown by row and column: rows are consumers, columns are providers.
- Matrix view adds the Rows / columns control so users can show all selected squads or only non-empty rows and columns.

If a dependency is visible in Sprint view but not in Matrix view, check whether both squads are selected in the saved view. This is the most common explanation when Sprint and Matrix do not show the same relationship.

Risk colours and counters

The current version uses deterministic risk rules rather than AI prediction. The summary counters and dependency colours use these statuses:

- Red / At risk: the dependency is overdue, blocked, carried over, unresolved, or has timing that may create flow-on risk.
- Amber / Watch: the dependency may affect upcoming work soon, or timing is close enough to need attention.
- Grey / Unknown: the app cannot calculate risk confidently because key date, sprint, link, mapping, or ownership data is missing or unclear.
- Green / Healthy: provider and consumer timing looks acceptable based on the current Jira data and app settings.

Grey is not automatically safe. It often means the team needs to improve Jira data before the dependency can be trusted.

The detail view should explain why a dependency has its current risk status. Use that explanation before deciding what action to take.

RAG filtering

The RAG counters can be used as filters. Typical use:

- Filter to red when the meeting needs immediate risk triage.
- Filter to amber when the group has handled urgent blockers and wants to prevent next-sprint issues.
- Filter to grey when the group is improving Jira hygiene or preparing for planning.
- Include green when the group wants a complete dependency picture.

Filtering changes what is displayed in the app. It does not change Jira data.

Dependency details

Open a dependency card or matrix cell when you need to understand why something is shown. Dependency details may include:

- Consumer issue: the issue that needs something from another squad.
- Provider issue: the issue expected to provide or unblock the work.
- Consumer squad/project.
- Provider squad/project.
- Jira issue key and summary.
- Direction label.
- Sprint context.
- Inferred date.
- Risk status.
- Risk or placement reasons.
- Links back to the underlying Jira issues.

Use the Jira links when data needs correction. Dependency Radar does not edit Jira issues, statuses, dates, or issue links.

Dependency types

The app normalises Jira issue links into two app-level dependency types:

- Blocks: one issue blocks another issue.
- Depends on: one issue depends on another issue.

The exact Jira link names and directions are configured by the client's Jira administrator or app administrator.

If a relationship appears backwards, missing, or under the wrong dependency family, ask the app owner or Jira administrator to review the dependency link mapping.

Direction filters

Direction filters apply to Sprint view:

- Incoming: selected squads are waiting on other teams.
- Outgoing: selected squads are blocking or providing work to other teams.
- Both: incoming and outgoing dependencies are shown together.

Matrix view does not need the same direction filter because direction is built into the layout: rows are consumer/dependent squads and columns are provider/blocking squads.

Saved views

Saved views control what a particular team group sees. Saved view settings include:

- View name and description.
- Visibility: new saved views are private to the creator by default. Choose Shared when the view should appear for everyone using the app.
- Selected squad projects. The current version treats one Jira project as one squad.
- Dependency direction filter: Incoming, Outgoing, or Both.
- Included dependency link types, within the app-level dependency mappings configured by the administrator.

The view dropdown groups saved views into default, private, and shared views. Private views are visible only to their owner. Shared views are visible to all users. Existing saved views created before private view support are treated as shared so teams do not lose access to established views.

Use private views for personal investigation, preparation, or a temporary subset of squads. Use shared views for recurring ceremonies and agreed delivery groups.

If a dependency is missing from the app, first check whether both squads and the relevant link type are included in the saved view.

Global settings

Global settings control how Jira data is interpreted across the app. They are maintained by the Jira administrator or app administrator. Global settings include:

- Dependency link mappings: which Jira issue-link types count as Blocks or Depends on, and which side of each link is the consumer or provider.
- Sprint calendar: the anchor/start date and sprint length used to generate the Sprint view columns. The default sprint length is 14 days.
- Matrix period: the period used by Matrix view, either a rolling period such as 12 weeks or a configured current-quarter date range.
- Done statuses: Jira workflow statuses that count as complete for dependency and risk calculations.

- Blocked or impeded statuses: Jira workflow statuses that should elevate risk when they appear on a dependency issue.
- Date fields: the preferred Jira date field and optional start date field used when placing dependencies and assessing timing.
- Risk thresholds: the red and amber slack-day thresholds used to classify dependencies as at risk, watch, unknown, or healthy.
- Stale-open dependency settings: whether old open dependencies should be detected, which age thresholds drive stale detection, and whether stale items should be shown, hidden, or isolated for review.

These settings do not change Jira issues. They only change how existing Jira issue links, statuses, dates, and projects are read and displayed in the app.

If a dependency has the wrong direction, colour, date, sprint placement, stale treatment, or matrix inclusion, the relevant global setting may need review.

Date calculations

Dependency Radar derives dates from live Jira issue data each time it loads the view. Jira remains the source of truth. The app does not write calculated dates back to Jira.

There are two related date calculations:

- Risk timing: compares when the provider issue is expected to be ready with when the consumer issue needs it.
- Sprint placement: chooses which sprint column should contain the dependency card.

Risk timing dates

For both the provider issue and the consumer issue, the app chooses the timing date in this order:

- 1 Configured preferred date field, if one is selected and populated on that issue.
- 2 Jira due date, if populated.
- 3 Sprint end date, if Jira exposes sprint data for that issue.
- 4 A planning-style fallback date field, if available. The app only considers non-system Jira date fields whose names look like planning dates, such as target, need, required, delivery, end, or finish dates.

The app ignores system and activity dates such as created, updated, last viewed, resolved, and status category changed dates. Those dates describe Jira activity, not delivery intent.

After the provider and consumer timing dates are chosen, the app calculates:

$\text{slackDays} = \text{consumer timing date} - \text{provider timing date}$

This means:

- Negative slack means the provider date is after the consumer need date.
- Low or zero slack is high risk.
- More slack is healthier.

If either side has no usable timing date, the dependency is grey because the risk cannot be calculated confidently.

Sprint placement dates

The sprint column is based on the issue shown in the selected squad row. For an incoming dependency this is usually the consumer issue. For an outgoing dependency this is usually the provider issue.

The app tries placement dates in this order:

- 1 Displayed issue start date, if configured or discoverable and populated.
- 2 Displayed issue preferred planning date, if configured and populated.
- 3 Displayed issue due date.
- 4 Displayed issue sprint start or sprint end date.
- 5 Displayed issue planning-style fallback date.
- 6 Linked provider or consumer issue timing, if the displayed issue has no usable date.

If the displayed issue has dates but those dates sit outside the visible sprint range, the dependency is not placed into the first visible sprint. It is omitted from the normal Sprint view until its row issue has a visible-window date, a usable inferred date, or explicit carried-over handling.

If no placement date can be found, the dependency is not forced into Current Sprint.

Sprint display filtering

Sprint view hides dependencies that are no longer useful for active sprint conversation.

The app removes a dependency from Sprint view when the consumer issue is done. It also removes a dependency when both the provider and consumer are done.

For open dependencies, the displayed row issue must map to the visible sprint window. Linked issue timing is only used when the displayed row issue has no usable placement date of its own.

If only the provider issue is done, the dependency is not treated as fully closed automatically. Instead, the app keeps it visible only when a consumer-side date falls inside the visible sprint window. Consumer-side dates include the consumer risk date, start date, due date, sprint start or end date, and planning-style fallback date. A current provider-side date by itself does not keep a provider-done dependency visible in Sprint view.

This means a provider issue completed in the current sprint will be hidden when the linked consumer issue still has only an old historic sprint or planning date outside the visible sprint window. Recent Jira activity on the consumer issue does not override this rule.

Stale-open dependency settings can hide old open dependencies when configured to do so. In hide mode, a stale dependency is hidden only when it does not have provider or consumer timing inside the visible sprint window. If either side has timing inside the visible sprint window, the dependency can remain visible for active planning conversation even when it is flagged as stale. Provider-done dependencies still have to pass the consumer-side visible-sprint rule above.

Common date use cases

- Preferred date field is set to End date and End date is populated: risk timing uses End date before Jira due date. Sprint placement still uses start date first if a start date is available, because start date better identifies where the displayed work begins on the timeline.
- Preferred date field is set to End date but End date is blank: the app falls back to Jira due date, then sprint end date, then a suitable planning-style fallback date. If none of those exists, the risk date is treated as missing and the dependency may become grey.
- No preferred date field is configured: the app starts with Jira due date, then sprint end date, then a suitable planning-style fallback date.

- Start date is blank but due date or sprint data exists: sprint placement falls back from start date to due date or sprint dates.
- Provider has a usable date but consumer has no usable date: risk is grey because the app cannot tell when the consumer needs the provider.
- Consumer has a usable date but provider has no usable date: risk is grey because the app cannot tell when the provider is expected to be ready.
- Provider is done and the consumer need/sprint date is inside the visible sprint window: the dependency can stay visible so the team can confirm the downstream work is still relevant.
- Provider is done but the consumer need/sprint date is historic or outside the visible sprint window: the dependency is hidden from Sprint view, even when the provider has a current sprint date or the consumer was updated recently.

Stale-open dependencies

Stale-open dependency detection helps teams separate active dependency risk from old Jira noise.

Stale detection applies only to open dependencies. If the dependency is closed for display because the relevant Jira issues are done, the stale-open filter is not the reason it disappears.

The stale filter is driven by Global settings:

- Past date days: the default is 90 days. The app looks at the provider and consumer risk/planning dates and uses the oldest date that is in the past. If that age is greater than the configured threshold, the dependency can be marked stale.
- Inactive days: the default is 60 days. The app looks at the provider and consumer Jira updated dates and uses the most recent update across the two issues. If neither issue has been updated within the configured threshold, the dependency can be marked stale.
- Default display: controls whether stale dependencies are shown with normal dependencies, hidden from the working view, or shown alone for cleanup.

In plain English, a dependency can be considered stale when it is still open but its planning date is old, or both linked Jira issues have had no recent activity.

There are two important exceptions:

- If an old planning date exists but one linked issue has recent activity, the app keeps the dependency out of the stale set because someone appears to still be working with it.
- If a dependency is stale but has provider or consumer timing inside the visible sprint window, hide mode keeps it visible so teams can still discuss it during active sprint planning.

Depending on the display mode, stale-open dependencies may be shown with normal active dependencies, hidden from the working view, or shown alone for cleanup.

The stale banner summarises how many stale open dependencies were found, how many are hidden in the current display mode, and the threshold values currently being used.

Use stale-only review when the team wants to clean up old links, close irrelevant work, refresh dates, or confirm that long-running dependencies are still real.

What good Jira hygiene looks like

Dependency Radar is only as useful as the Jira data it reads. Teams get the best results when they:

- create Jira dependency links as soon as cross-squad dependency is known,
- use agreed link types consistently,

- keep provider and consumer issue statuses current,
- populate the configured preferred date field or Jira due date,
- maintain sprint data where sprint placement matters,
- close completed dependency work,
- remove or update old links that no longer represent active delivery risk,
- avoid using activity dates as planning dates,
- agree which Jira projects represent squads in the saved view.

If the app shows too many grey, missing, stale, or unexpected dependencies, the next improvement is usually Jira hygiene rather than app configuration.

What to do if something looks wrong

Use this checklist before raising a defect:

- Check that the Jira issues are linked correctly.
- Check that the relevant dependency link has actually been created in Jira.
- Check that the dependency link type is included in the app configuration.
- Check that the link direction matches the app-level Blocks or Depends on mapping.
- Check that both squads are selected in the saved view.
- Check that the saved view includes the relevant incoming, outgoing, or both direction filter.
- Check that Jira dates used by the app are populated and maintained.
- Check that the sprint calendar is configured correctly.
- Check whether the issue has enough date or sprint information to place it.
- For Matrix view, check that the matrix period is configured correctly.
- For Matrix view, check that both squads involved in the dependency are selected in the saved view.
- Check whether stale-open settings are hiding or isolating old dependencies.
- Ask the app owner or Jira administrator to review the saved view and global settings.
- Raise a support question or defect when the Jira data and settings look correct but the app still shows the dependency in a way that cannot be explained from the detail reasons.

Example user scenarios

Scenario 1: Preparing for scrum-of-scrums

A scrum master opens the shared saved view for the delivery group. They filter to red and amber dependencies, check which squad rows have the most pressure, then open Matrix view to see whether the same provider squad is involved in multiple risks. They bring only the top relationships into the meeting.

Scenario 2: A dependency is missing

A squad lead expects to see a dependency but it is missing from Sprint view. They check whether the Jira issue link exists, whether the link type is mapped, whether both squads are selected in the saved view, and whether the displayed row issue has a usable date inside the visible sprint window.

Scenario 3: Too many grey dependencies

A delivery manager sees a high grey count. They filter to Unknown, open several dependency details, and discover that provider or consumer dates are missing. The follow-up action is to improve Jira dates, not to treat the dependencies as low risk.

Scenario 4: Matrix shows a concentrated relationship

Matrix view shows one consumer squad depending heavily on one provider squad, with a red cell. The facilitator opens the cell, reviews the dependency list, and asks the two squads to confirm ownership, target dates, and the next action in Jira.

Scenario 5: Cleaning up old open links

The app owner uses stale-open settings to isolate old open dependencies. Teams review the list and either close completed Jira work, refresh dates for active dependencies, or remove links that no longer represent real delivery risk.

What the app does not do

Dependency Radar currently does not:

- edit Jira issues,
- create Jira dependencies,
- maintain a separate dependency database,
- replace Jira issue links,
- replace Advanced Roadmaps / Plans,
- model plan hierarchy or scenarios,
- run AI prediction,
- decide delivery ownership,
- guarantee that Jira data is correct,
- show dependencies to squads outside the selected saved view in the main matrix,
- act as a historical trend report,
- provide a dashboard gadget in the current version.

These limits are intentional. The app is designed to stay read-only and focused on dependency visibility, delivery risk, and planning conversation.

Matrix view limitations

Matrix view is deliberately narrower than Sprint view. Matrix view only shows dependencies where both sides of the relationship are in the selected saved view:

- the consumer squad must be selected,
- the provider squad must be selected,
- the link type must be included,
- the dependency must fall inside the configured matrix period or meet the unresolved/missing-date inclusion rules.

Sprint view can be broader for a selected squad. If a saved view includes Squad A, Sprint view can show Squad A's incoming or outgoing dependencies even when the linked squad is not part of the saved view. That makes Sprint view better for inspecting a selected squad's dependency workload.

Matrix view is better for comparing relationships inside a defined group. It is not a complete map of every external dependency touching that group unless the external squads are also selected.

Practical implication: if Matrix view looks quieter than Sprint view, it may be working correctly. Add the missing provider or consumer squad to the saved view when the group wants that relationship represented in the matrix.

Current limitations

- Jira Cloud only.
- One Jira site per installation.
- One squad equals one Jira project.
- The current version assumes aligned sprint cadences for selected squads.
- No Advanced Roadmaps integration.
- No dependency creation inside the app.
- No Jira issue editing inside the app.
- No dashboard gadget in the current version.
- Matrix view is an aggregate discussion view, not a dependency history or trend report.

Support and contact

For questions about installation, configuration, or how to interpret the dependency views — or to suggest improvements — email support@aboutthinking.org.

Dependency Radar does not replace Jira, Advanced Roadmaps, or your existing delivery process. It gives teams a focused dependency lens over the Jira work they already maintain.