Plans for non-tech teams: a semantic dissection from a Content Designer

What's this about?

As part of the work around Plans for non-tech teams, we're exploring how to adapt Plans to be more accommodating to non-technical teams (I'll refer to these as ***NTTs***).

This page focuses on the content experience, identifying ways we can modify our language structures and overall content strategy to be more accommodating of less technical audiences from non-software backgrounds.

This work is independent from [other work] which is working to redefine common Jira terminology and items. Here, I'm referring to how we frame features and define actions within Plans specifically.

Part 1: Contextually-defined terms versus lingo (and why lingo's bad)

Atlassian's Content guidelines state that we "Avoid using jargon" largely because jargon relies on everyone having the same definition in order to work.

Jira, and Plans in particular, relies heavily on Agile-specific terminology – dare I call it jargon. Using this language, we've been able to make very precise and complicated planning tools for those who understand this vernacular. However, it excludes customers who don't know this terminology – a common trait among those who plan for non-tech teams (NTTs).

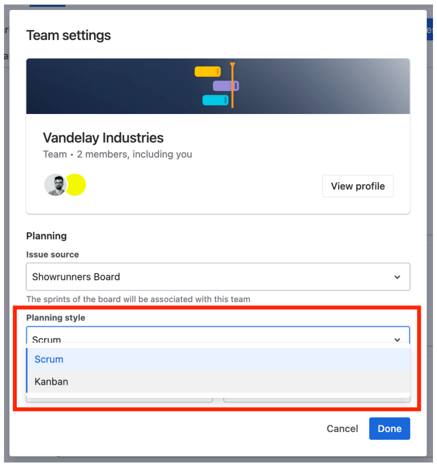
We need to move away from Agile-specific terms and instead define concepts and features contextually. This will make concepts approachable to NTTs while not alienate our existing Agile user base. This looks like:

1. reframing the planning experience to define concepts through specific attributes, value-props, or outcomes while also
2. grounding content in the immediate context of the screen

For a quick example, let’s look at the team configuration screen inside of Plans.

Today, when a planner sets up a plan, we ask them whether a team is **scrum** or **kanban** →

If the plan is being made by someone with no understanding of Agile, this prompts questions like “What’s scrum?” and “What’s kanban?” Naturally, an NTT lead looks up definitions of these words, and they’ll find something like this (taken from the Atlassian Agile Coach):



Team settings

Vandelay Industries

Team • 2 members, including you

View profile

Planning

Issue source: Showrunners Board

The sprints of the board will be associated with this team

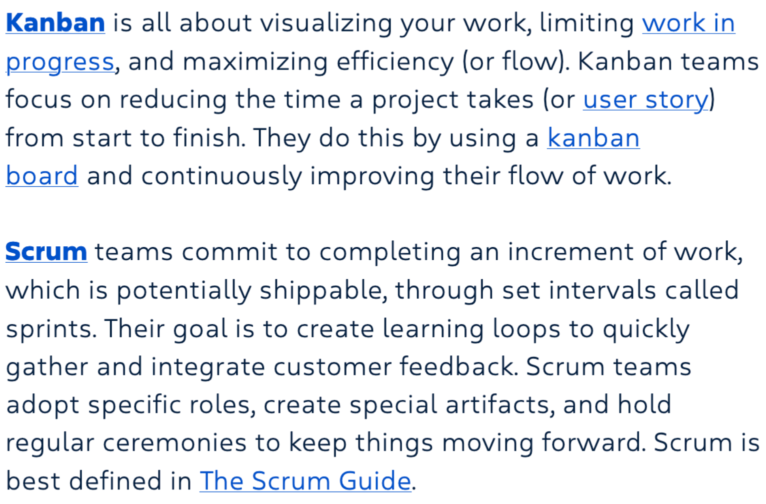
Planning style: Scrum

Scrum

Kanban

Cancel

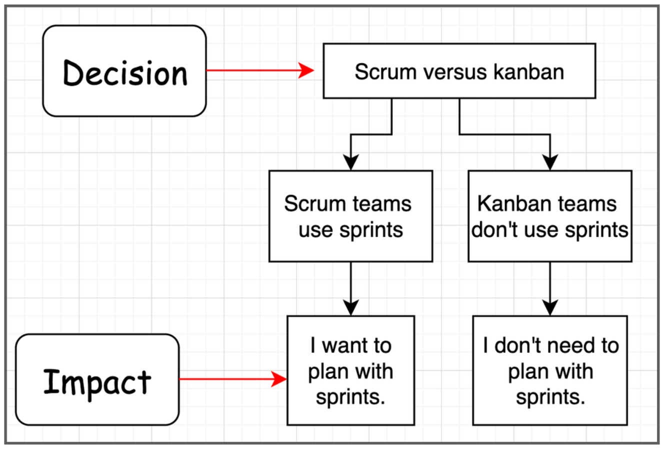
Done



Kanban is all about visualizing your work, limiting work in progress, and maximizing efficiency (or flow). Kanban teams focus on reducing the time a project takes (or user story) from start to finish. They do this by using a kanban board and continuously improving their flow of work.

Scrum teams commit to completing an increment of work, which is potentially shippable, through set intervals called sprints. Their goal is to create learning loops to quickly gather and integrate customer feedback. Scrum teams adopt specific roles, create special artifacts, and hold regular ceremonies to keep things moving forward. Scrum is best defined in The Scrum Guide.

The question in the UI is asking about how this team plans with specific regard to the way their workflows are set up — ie, do they use sprints? We ask which broad methodology are they using with the expectation that they know which label applies to their situation. We don’t tell them how to spot the difference.


Decision
Scrum versus kanban
Scrum teams use sprints
Kanban teams don't use sprints
Impact
I want to plan with sprints.
I don't need to plan with sprints.

Without a full understanding of the connotation of these terms, NTT users disengage from the product.

"I'LL HAVE THE IKE AND TINA TUNA."
"PLATE OR PLATTER?"
"I DON'T UNDERSTAND THE QUESTION, AND I WON'T RESPOND TO IT."

This can be a recipe for frustration to users who already recognize the value of Plans (via marketing demos, or other means); we've enticed the customer with benefits then obscured it by undefined terminology.

How we improve

To make an experience that works for all users, we need to close the gap between what's being asked, and the impact of that decision. For this, we need to:

1. reframe the question to more directly correlate to the impact and

2. ground it in the context of the flow

With that said, we might rewrite the prompt to ask:

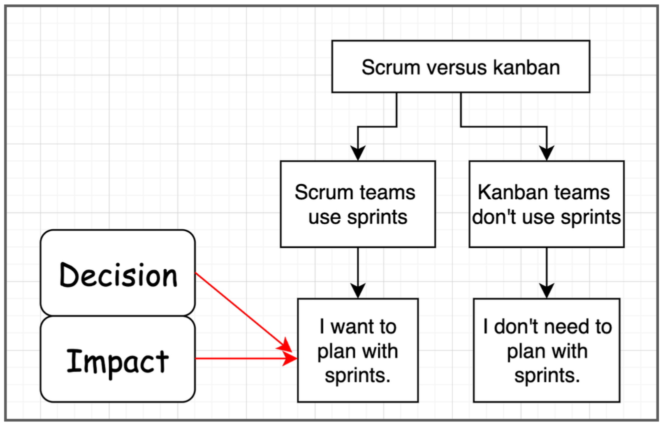
Does this team use sprints?

This reframing shifts the dialogue from:

"I don't know what this is asking"

to a more certain

"That isn't what I want."


Scrum versus kanban
Scrum teams use sprints
Kanban teams don't use sprints
Decision
Impact
I want to plan with sprints.
I don't need to plan with sprints.

Which is fine! The point of a well-defined interface like this isn't to get all planners to use all features; it's about empowering users to do what's right for them.

NTTs might not want to use all the bells and whistles that tech teams use in Plans, but at least they'll know enough to make an informed decision. At the same time, those who know what scrum and kanban are will recognize this as being an Agile-specific concept and make the correct choice.

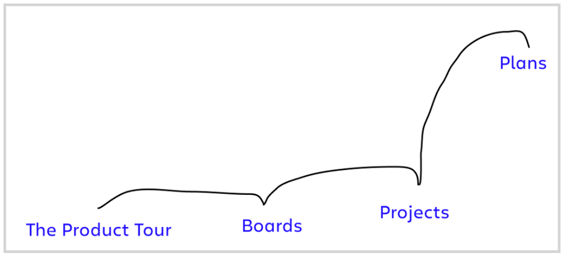
Part 2: Learn as you go with Progressive Disclosure

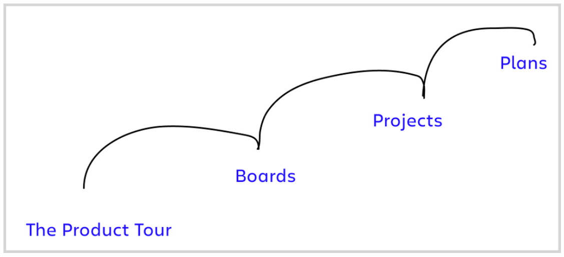
The idea behind progressive disclosure is that we teach customers about concepts while they use and explore the product.

But our current use of language in Plans is the opposite of progressive disclosure.

We've leaned heavily on Agile lingo in the past because leveraging connotatively-laden terms like "scrum" and "kanban" greatly reduces our design scope. But we do so with the expectation that customers know the terminology – which excludes NTTs and new customers.

In order to incorporate progressive disclosure principles into our designs, we need to think about where a concept is introduced, how it's framed, then consciously leverage previous uses to build customer's understanding of concepts and functionalities. In a product like Plans, that might be in other parts of Jira.





not good *more good*

As a result, a well-designed, progressively-disclosing experience will:

1. build on top of pre-existing knowledge
2. use the customer's data and experience to demonstrate the value of the larger concept

To demonstrate how we could better leverage progressive disclosure in Plans (nay, Jira as a whole), let's look at capacity planning.

Capacity planning is a way for planners to track the amount of planned work in order to gauge whether the amount of work is feasible to get done in that timeframe. It uses iteration lengths and issue estimates which both live outside of Plans. Yet we don't mention capacity anywhere in Jira at large.

Instead, the concept is presented as a standalone concept within Plans, devoid of the powerful context and connections that comprise its key selling points. This framing rewards those who know Jira well enough to link the two concepts but alienates NTTs and new customers who have to learn this new concept and how to use it.

How we improve

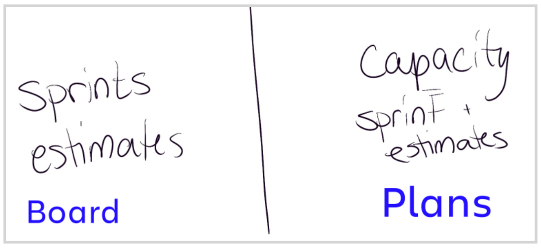
Given that capacity uses two existing concepts in Jira, we can:

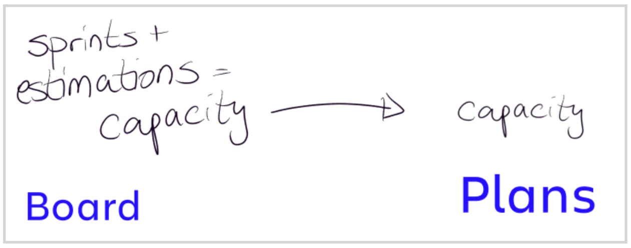
a) introduce the concept of capacity planning before
   1. they ever need to use it in Plans
   2. they upgrade (upsell opportunity)

b) use their own sprint and issue estimates to demonstrate how capacity works

c) reduce the onboarding learning curve when they use Plans for the first time

By priming customers with this language before they get into Plans, we can spend less time explaining the concept and jump straight to its benefits.





us now *us if we stop being silly*

Making this big of a change presents us with a lot of options for how to execute. As before, the ideas in this section are for demonstrative purposes only and are taken from my previous work: Does Teams still deserve its own tab?

To this end, we could:

• reference 'capacity planning' in the board settings page

We introduce the idea of "capacity" on the board, predominantly in the settings and the sprint information page of a Team's board.

• add a Capacity field on the board

We let team planners add a value for a target amount of work per sprint and label it Capacity. For non-Premium customers, it'd have little use aside from getting the concept into planners heads, but for those on Premium plans, this is information that could be defined by the team, removing the requirement that planners configure this information

• automatically ingest sprints from boards

Instead of making planners set independent values for capacity in Plans, we ingest the data from the associated boards: when the sprints start/end, the team velocity (which we're already working on automating).

What these options accomplish/demonstrate:

1. They allow us to leverage an existing concept that customers already understand (issue estimates) to demonstrate how that scales up to Capacity planning.
2. Primes customers to the concept before they use it. For existing Premium users, this means they'll have a better understanding of the concept before they land in Plans and non-Premium customers will be exposed to the concept before upgrading.
3. Inside of Plans, we can jump straight into the insights of the capacity planning tool and spend less time explaining what capacity is.

Any one of these would reduce the amount of learning and the time to value for customers once they enter Plans.

Part 3: Customer built complexity

As I said before, progressive disclosure is all about layering information over time to construct a concept for customers. While the goal for all audiences is the same (to plan goodly), the experience of doing so isn't. The features planners need, the information their plan shows, and how changes are reflected are all dependent on how their organization plans, not whether they see value in the tool.

The point of a well-designed interface isn’t to get all planners to use all features; it’s about empowering users to make the right choices for their situation.

To this end, we should design for simplicity, then allow customers to layer on complexity when they indicate they're ready for use it. If a feature has little or no value to a customer on a given planning path, then we shouldn't show it to them in the first place.

If we execute on this design principle, we can:

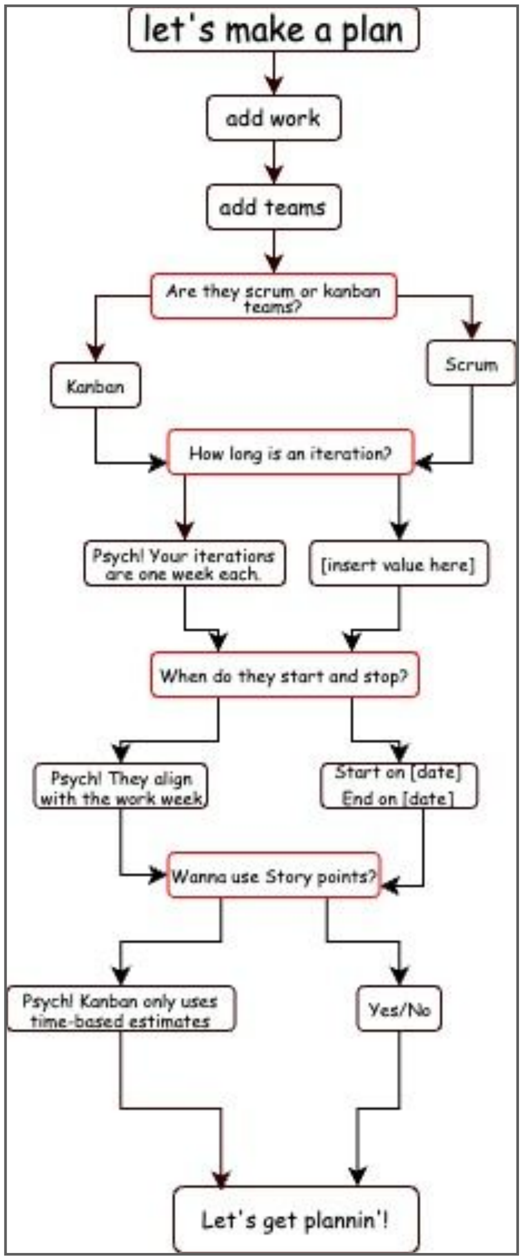
- lower the cognitive load of a customer learning how to use the product for the first time
- only show features that are relevant to their planning experience and from which they'll derive value
- afford them the room to enable more features as their planning becomes more mature

To demonstrate this, let's look at a flow chart for configuring kanban and scrum teams.

In this flow, a planner is setting up a team in their plan. For capacity planning, we need to know:

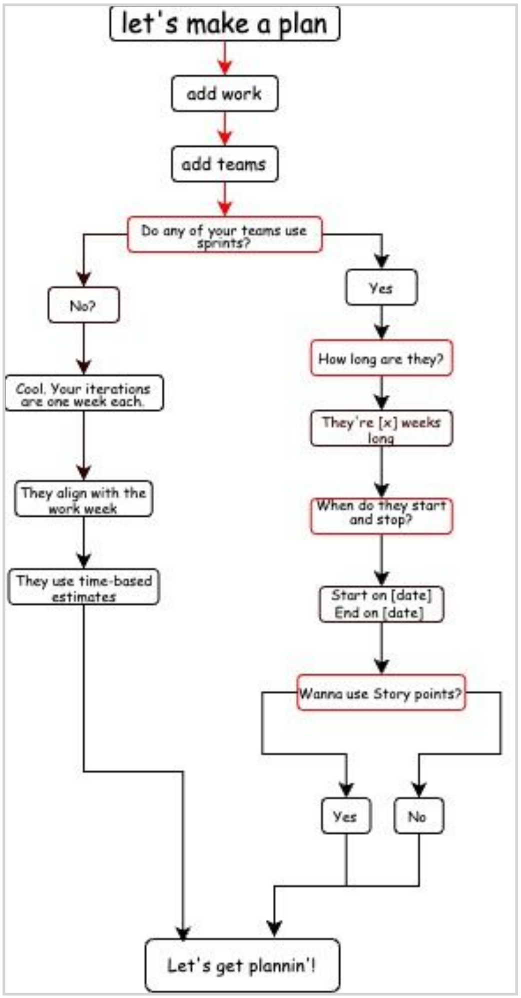
- scrum vs. kanban
- iteration length
- when those iterations happen
- estimation unit

But for kanban users, only one of those decisions actually matters (kanban planners can't change the iteration length, when they happen, or estimation units). Yet the kanban planner needs to go through all of these steps to get to the same end goal because they're in a flow designed for a different audience.


let's make a plan
add work
add teams
Are they scrum or kanban teams?
Kanban
Scrum
How long is an iteration?
Psych! Your iterations are one week each.
[insert value here]
When do they start and stop?
Psych! They align with the work week
Start on [date]
End on [date]
Wanna use Story points?
Psych! Kanban only uses time-based estimates
Yes/No
Let's get plannin'!

How we improve

Instead of thinking that every experience needs to work for every user, we should work to identify when we can simplify the experience while also giving customers the chance to add complexity as their planning practices mature.


let's make a plan
add work
add teams
Do any of your teams use sprints?
No?
Yes
Cool. Your iterations are one week each.
They align with the work week
They use time-based estimates
How long are they?
They're [x] weeks long
When do they start and stop?
Start on [date]
End on [date]
Wanna use Story points?
Yes
No
Let's get plannin'!

In contrast to the example above, if a planner chooses kanban instead of scrum, we can bypass most of the confusing and irrelevant decisions that have no impact.

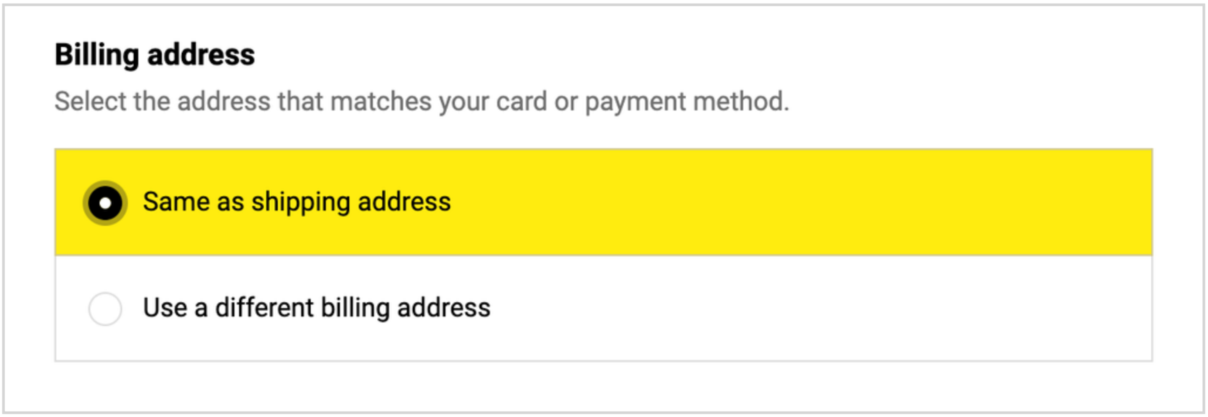
Now instead of four decisions (three of which give the illusion of choice), kanban planners now have one. This decision also reflects outward in that we can hide all of the scrum-only information and concepts from their plan.

This concept can be applied to Plans at large:

- if they have no dependencies or releases in their plans, do we need to show the tabs?
- do we need to show all of the settings all of the time in the Plan settings page?
- can we hide warnings that aren't applicable or aren't currently in use?

Or, on a larger, philosophical note, how do we shift away from designing complexity and then abstracting out to meet simple customers and and instead design experiences to be simple, then layer complexity on top?

One minute but real example of this is when you're buying something online, and companies let you use your shipping address as your billing address.



Billing address

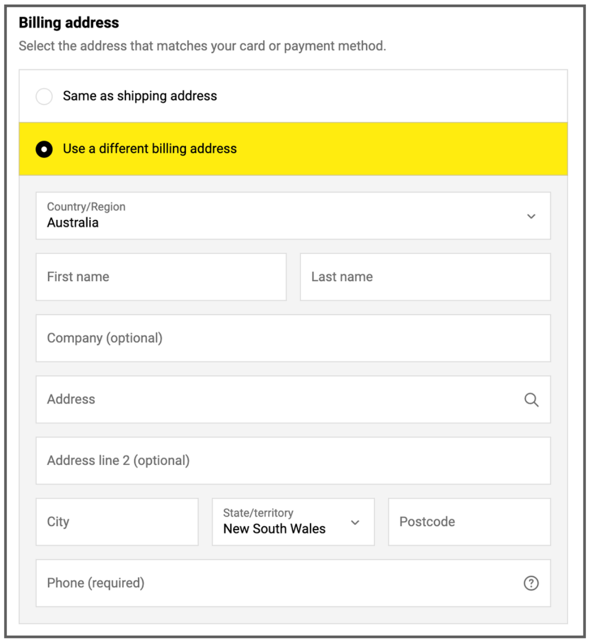
Select the address that matches your card or payment method.

[✓ Same as shipping address]

Use a different billing address

The default experience is designed for simplicity – using the same address. The more complex customer path is for those who are shipping to a different place than where they live (maybe they'll send their new iPhone to the office instead of their house).

However, this complexity is a detour only those who self-select into it will see.


Billing address

Select the address that matches your card or payment method.

○ Same as shipping address

● Use a different billing address

Country/Region: Australia		
First name	Last name	
Company (optional)		
Address		
Address line 2 (optional)		
City	State/territory: New South Wales	Postcode
Phone (required)		

Conclusion

As I've been thinking about this piece, I've come to the realization that we're progressing past the easy wins for improving Plans for a broader audience. We've sanded down most of the rough edges and we're left with more substantial changes that aren't going to be quick, nor will they be easy.

But that's a half-empty way of thinking about this. The converse is that, if we're willing to take big swings at this, we can:

- empower customers to make better decisions and feel more in control of their planning experience
- reduce complexity by hiding features from customers that they won't find valuable
- create pathways for customers to grow as their planning rituals become more mature