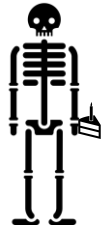
As Simple As Possible…How to Create Smaller Stories and Tasks to Deliver Value Quickly

David Frink



David Frink

Developer -> Dev Manager -> Agile Coach/Release Train Engineer

Ipreo, Teradata, SciQuest
PSM, ICP-ACC, ICP-ATF, ICP
david@dfrink.com

Why Simpler Stories?

- Every story is a guess
- Short feedback loops test our guesses
- Deliver value sooner
- Value = value to customer
- Value = learning for team

Working First, Awesome Later

Basic Philosophy for Finding Simplicity

- **Working** first, awesome later
- **Possible** first, easy later
- **Build** first, polish later
- **Learn** first, ship later

80% of the value comes from 20% of the effort…do that 20% first

Agenda

- Form Teams
- Series of exercises using a common experience (restaurant) to demonstrate splitting/simplification techniques

Key things to note:

- At the end I'll share a link to notes and resources
- We've got a lot to cover, so let's get started

Forming Teams

Set-up

- Teams of 3-4 people
- You'll need space to put sticky notes
- Raised hand means exercise is over

Instructions

- Form your team (3-4)
- Introduce yourselves
- You're creating a restaurant
- Write the name of your restaurant on a sticky note

Exercise 1 – Big Project – Opening a Restaurant

Set-up

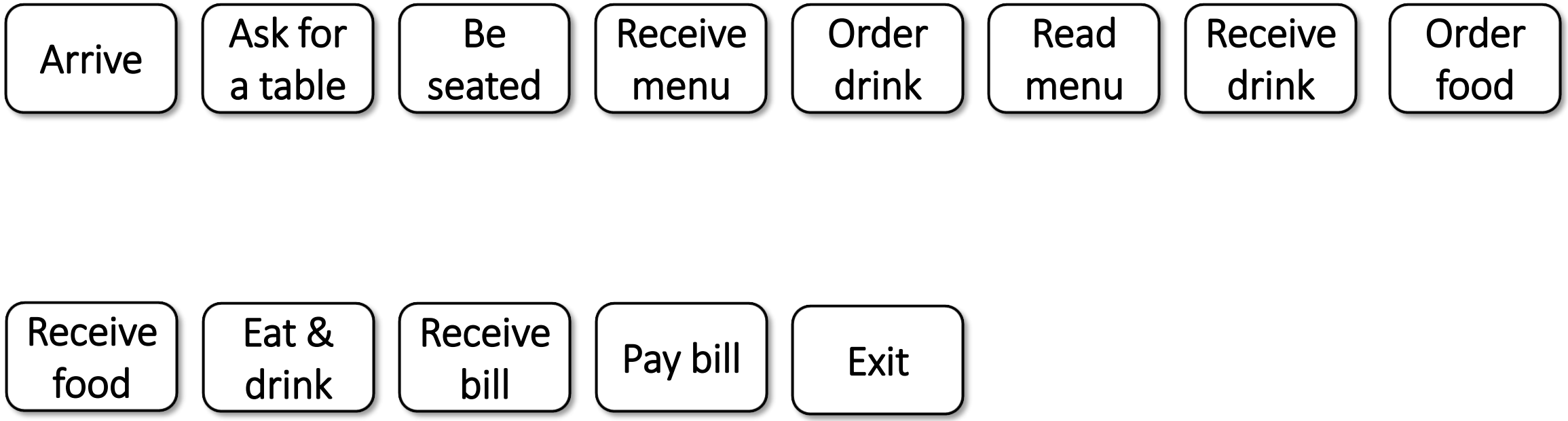
- Opening first restaurant (sit-down)
- Don’t worry about the building, hiring staff or what happens in the kitchen. Only focus on what the customer sees.
- Want to understand customer experience

Instructions

- Create the customer’s experience, one sticky per event
- Arrange from left to right/start to finish
- Start with verbs: “Ask for a table”

Exercise 1 Debrief

Exercise 1- Customer's Experience



Exercise 2 – Restaurant MVP (minimum viable product)

Set-up

- Before you open you need an MVP
- For each key step, identify the key things you need to support before you can do your “grand opening”
- Suggestions – party size, drink options, menu choices, payment methods

Instructions

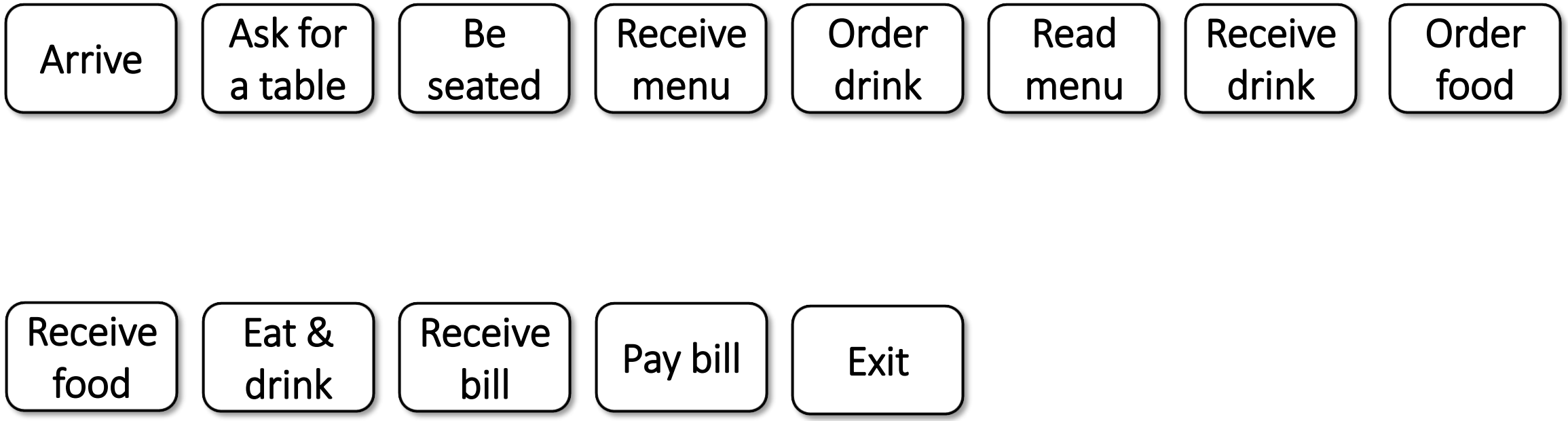
- Add one sticky note per MVP feature below each key step
- Won’t have much time, so make sure to get 2-3 options in each step (ignoring ones like “Eat Food”)

Exercise 2 – Ideas if you're stuck

- Host/hostess
  - Small Parties
  - Large Parties
  - Reservations
- Drinks
  - Water, Soft Drinks, Tea, Beer, Wine, Kombucha
- Food
  - Appetizers…
  - Entrees…
  - Desserts…
- Payment Methods
  - Cash
  - Visa
  - MasterCard
  - Amex
  - Diner's Club

Exercise 2 Debrief

Restaurant Workflow



Exercise 3 – Happy Path

Set-up

- You have your flow and MVP, but it will take too long to do all of it
- You've never run a restaurant and want to test your flow works first
- Need to test each component (host/hostess, waitstaff, kitchen, payment)
- Need to test connectivity/handoffs
- Want to be sure your restaurant works before you make it more complicated

Simple steps to a “Happy Path”

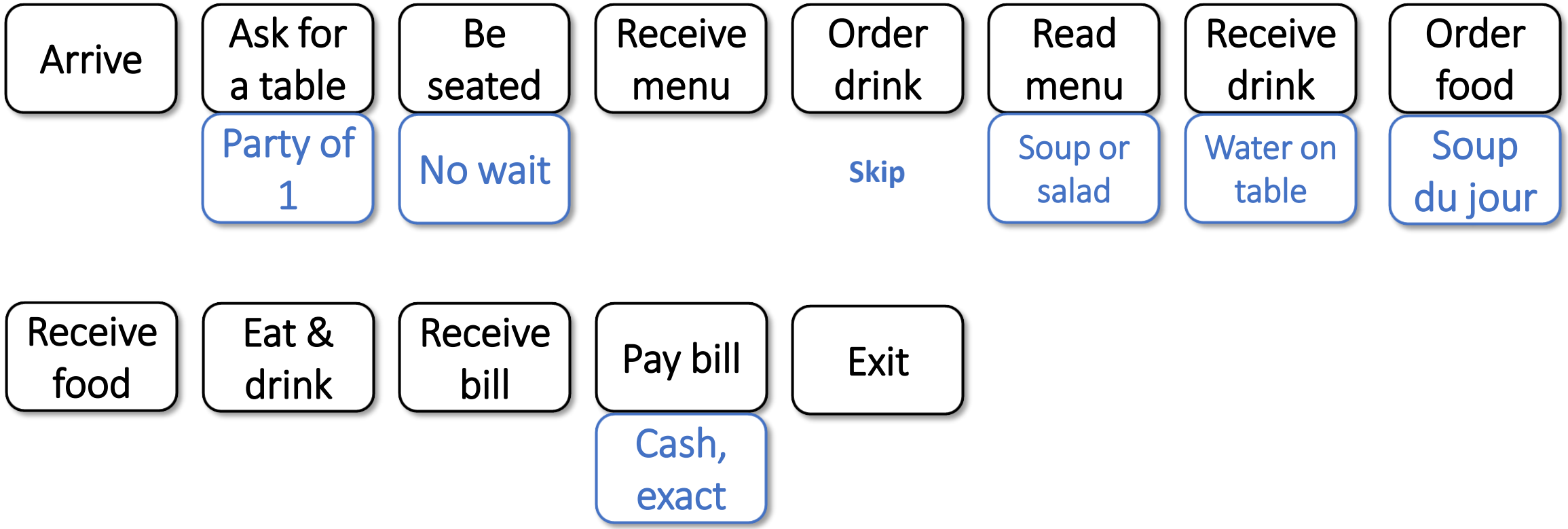
- INVEST, specifically:
  - Independent
  - Valuable
  - Small
- Variations in complexity – Easy/Hard
- Variations in frequency – Always/Seldom
- Make assumptions
- Reduce decisions

Instructions

- Find the “Happy Path” - Identify the simplest way to get through each step
- Pick one small feature to support per step
- Write this on a new sticky note below each step
- “Party of 1”

Exercise 3 Debrief

Exercise 3- Happy Path



Patterns for early learning (80/20)

- MVP is rarely “minimum” – focus on learning
- Test the system first, then worry about customer value
- Happy Path first – Pretend it is easy
  - Make assumptions (all users are the same and they don’t make mistakes)
  - Skip configuration
  - Skip interfaces
- Split on variations in data
- Split on variations in complexity
- Build the thing with the fewest options first (reduce choice)
- Use INVEST to test your stories for splittability
- And…

It’s Alive! (and has cake)

- **Walking Skeleton**
  - Backbone with just enough meat to make it move
  - Confirms all components exist and can communicate
  - You can choose where to invest next

- **Thin Slice**
  - Each slice of cake hits all the layers
  - If you can create a thin slice, you can come back and make the slices bigger later

But wait, there's more…it slices, dices and even works on technical stories and tasks

- Thin Slice is the opposite of a "technical story"
- Not:
  - (0% value) Story 1 – Database with 30 fields
  - (0% value) Story 2 – Business layer with 30 fields
  - (0% value) Story 3 – API with 30 fields
  - (100% value) Story 4 – UI with 30 fields

But wait, there's more…it slices, dices and even works on technical stories and tasks

- Instead:
  - (10% value) Story 1 – 3 fields functional in: UI, API, Business Layer, Database
  - (33% value) Story 2 – 7 fields functional in: UI, API, Business Layer, Database
  - (100% value) Story 3 – 14 fields functional in: UI, API, Business Layer, Database*
- And for tasks, each layer can be a task…because it is something you can get feedback on (code review, early look, etc.)

* Only 24 fields because by iterating, you figured out you didn't need 6….That's 100% of the value at a 20% savings to you!!!

Exercise 4 – Find the core value

Set-up

- Step back from your flow and think about which steps provide the most value
- There are 2-3 steps that are most risky…without them you don’t have a restaurant

Instructions

- Identify the 2-3 steps that are most valuable/risky
- Once you’ve identified them, think of ways to test if they work without your whole workflow

Exercise 4 Debrief

What is your RAT (Riskiest Assumption Test)?

Arrive

Ask for a table

Be seated

Receive menu

Order drink

Read menu

Receive drink

Order food

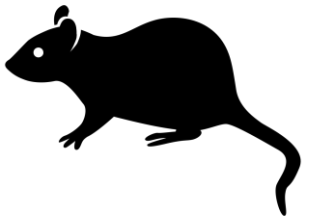
Receive food

Eat & drink

Receive bill

Pay bill

Exit

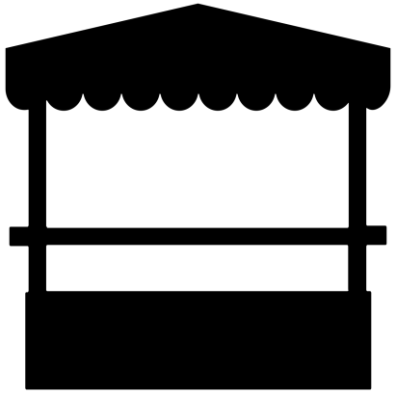


What is the riskiest assumption?

Cook good food

Get paid

How to test this assumption with less effort?



The Only 3 Problems You Have

- Can I get it working?
- Does it add value?
- Everything else

Until you solve the first 2, here's what doesn't matter:

- Usability
- Performance
- Scalability
- Edge cases
- Availability

(The stuff on the right does matter and you need to account for it…but unless you know you're solving the right problem, it can be wasted effort.)

Disclaimer: If your team doesn't respect DoD or your organization doesn't have the discipline to learn first then finish the feature the right way, these techniques may cause trouble

Patterns for early learning (80/20)

- MVP is rarely “minimum” – focus on learning
- Test the system first, then worry about customer value
- Happy Path first – Pretend it is easy
  - Make assumptions (all users are the same and they don’t make mistakes)
  - Skip configuration
  - Skip interfaces
- Split on variations in data
- Split on variations in complexity
- Build the thing with the fewest options first (reduce choice)
- Use INVEST to test your stories for splittability
- Walking Skeleton/Thin Slice 
- Find the RAT 
- What’s your food truck? 

Key Questions to find simplicity

- “Is this a problem we have right now?”
- “What is our biggest risk?”
- “How long can we get away without solving this problem?”
- “What if that wasn’t configurable at first?”
- “What will users want 90% of the time?”

Working First, Awesome Later

(assuming this approach won't lead to you shipping half-done stuff and moving on)

Further Reading

www.dfrink.com/simple

Thanks/Questions

www.dfrink.com/simple