Elephant Carpaccio exercise

Facilitation guide

Henrik Kniberg & Alistair Cockburn, 2013-07-11

What is this?

The Elephant Carpaccio exercise is a great way for software people to practice & learn how to break stories into really thin vertical slices. It also leads to interesting discussions about quality and tech practices.
The exercise was invented by Alistair Cockburn. We've cofacilitated it a bunch of times and we encourage people to run it everywhere.
This is a detailed (shu-level) facilitation guide based on how Alistair runs it plus some minor adaptations from Henrik.
The exercise takes 90-120 minutes, and scales well. We normally do it with 10-20 people but have also done it with groups up to 30.

Timing

2 hours is best, 1.5 hour works but feels a bit rushed.

15-20m discussion about user stories
20-30m breaking down backlog
40m coding
15-20m debrief

It's possible to skip the coding part & just practice creating the backlog. But it takes away much of the fun, and you also miss out on interesting code-quality discussion at the end.

Preparation

Print this handout for everyone (or write the bottom half of it on a whiteboard/flipchart).

Make sure half of the participants have a laptop with a working dev environment.
Less is OK, but at the very least every third participant should have one.
Make sure there are power strips in the room.
Internet isn't really needed for the workshop, but it is helpful to be able to look stuff up while programming.

Running the workshop

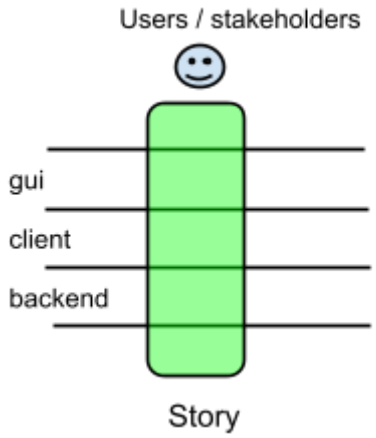
Intro

Purpose of this workshop is to learn to split stories really small..

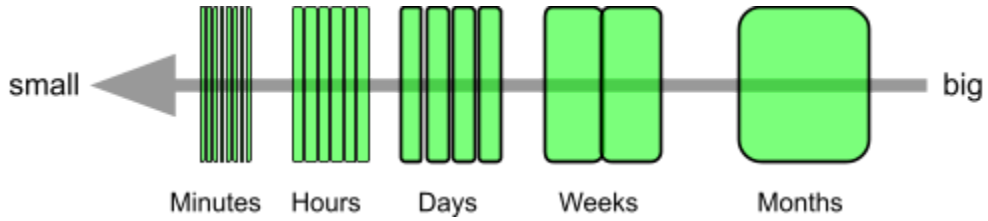
Why split stories (optional)

This part is only needed if you haven't already discussed the value of splitting stories

Story = vertical, testable, user-valuable. Cuts across multiple architectural layers.



Story slicing = making thinner stories (but still vertical)

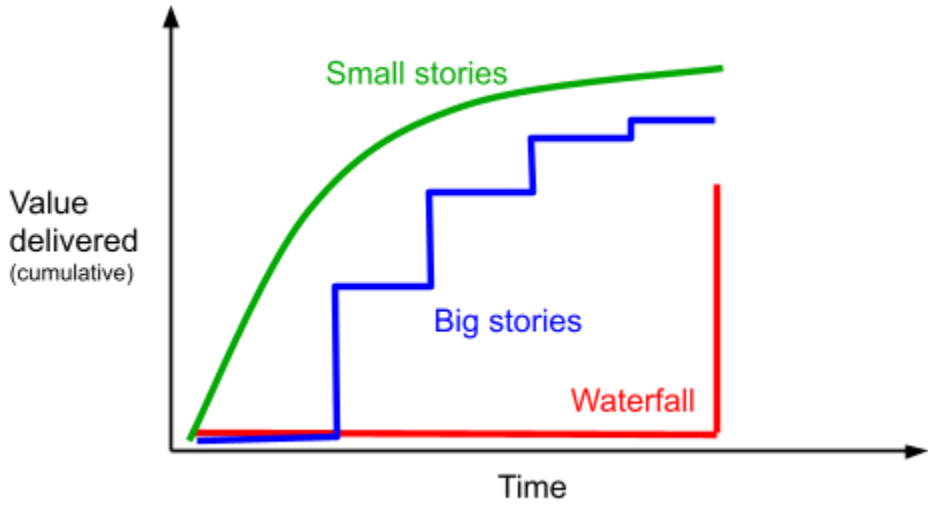


How big are your stories? tasks? commits? (mark on size-line)
Target: Story = a few days. Task = a few hours. Commit = several times per hour.
Why split stories? Discuss in pairs. (ask each pair to report, ask what's missing, add any of the below that are still missing).

Learn faster. Deliver more often. Happier stakeholders. More in-sync with other people & teams. Better prioritizations. Better product earlier. More

business options. Less risk (less time “underwater”). Sense of velocity. Easier planning.

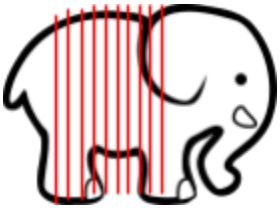
Draw value curves for waterfall, big stories, and small stories. Discuss. Why does the “small stories” curve end up with higher cumulative value?



Decision of “how small” should not be limited by “we can’t split this story”. In this workshop we will practice by exaggerating. We will make stories so tiny that anything you do today seems big in comparison.

What we will do

Build a simple application in 40 minutes, divided into 5 iterations x 8 minutes.
Most people would build this app in 2-3 slices, we will do it in 15-20.
Elephant carpaccio = very thin slices, each one still elephant-shaped. Together they form the whole elephant.



The product

Close your laptops! (or people will be fiddling with their environment instead of listening)
(refer people to the handout).
We will build a retail calculator. It is a runnable application with user interface, three inputs and one output.

Use any programming language. Interface could be console, command line, web, gui, whatever.

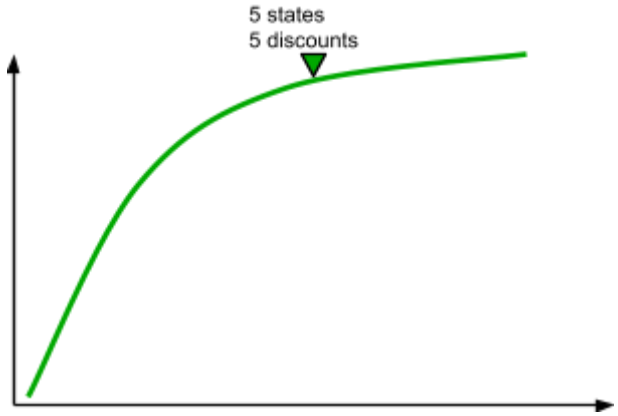
Three inputs:

- How many items
- Price per item
- 2-letter state code

Output: total price of the order. Give a discount based on the total price, then add state tax based on the state code and discounted price.

Priorities

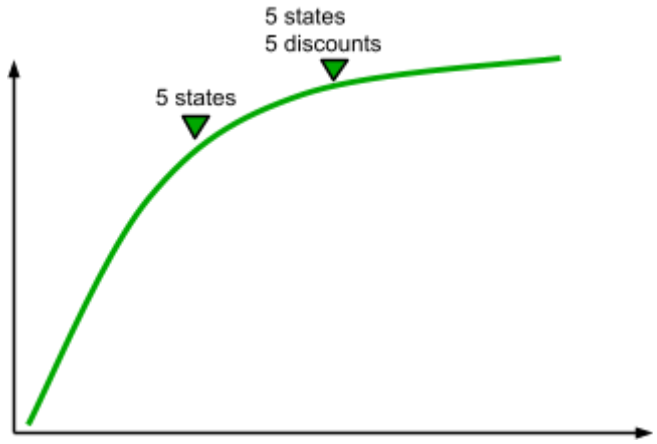
I will illustrate priorities on this **value curve.**
Target is 5 discounts, 5 states.



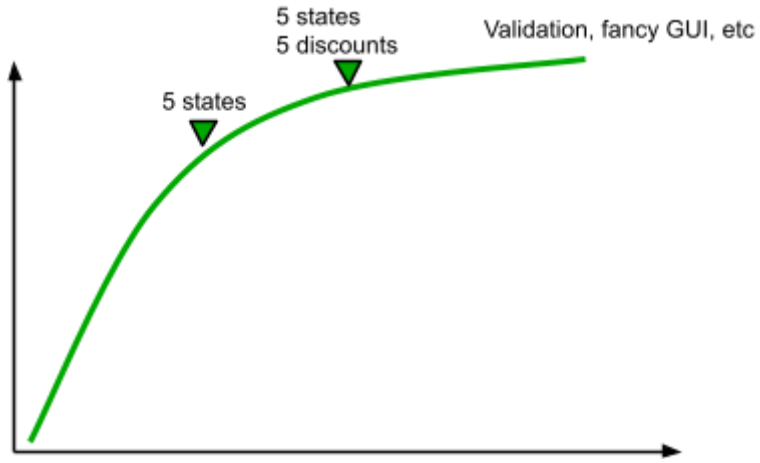
You will break this into **10-20 slices.**

A slice is only valid if it has a UI, input & output, and is visibly different from last slice.

I want **5 states before doing anything with discounts**. Why? (so we can deploy! State tax is a legal requirement, discount is not)



Validation & fancy GUI comes *after* 5 states & 5 discounts. Don't spend any time on that before!



No embellishing! Practice on building the highest-value thing first, all the time. For example, before you have 5 states, don't waste a single keystroke on discount-related code!

I **don't care how far you get on this curve**. The important thing is that you practice microslicing. If you get all the way to the end with only 3 slices, you've wasted your time and missed the point of this exercise.

Create the backlog

Split into groups of 2 or 3 (2 is better). Each group should have at least one programmer with a working dev environment.

10 minutes: Write your backlog on paper (laptops still closed).

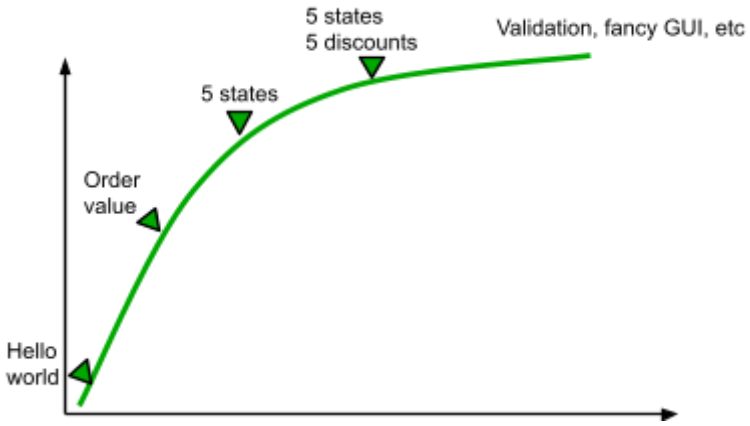
- Write 10-20 demo-able user stories ("slices") that will take you from nothing to 5 states & 5 discounts.
- Valid slice guideline:
  - Implementable (including user interface) in 2-6 minutes.
  - Noticeable different from last slice
  - More valuable to customer than last slice (exception: for first couple of slices, focusing on reduced risk is OK).
- No slice is just a mockup or UI, a data structure, or test case.

What is your first slice?

- (anything bigger than hello-world or echo-input-to-output is too big)
- Discuss risk-reduction value of implementing & "delivering" hello world as first slice. Value = customer value + knowledge value
- Discuss value of quickly building a "walking skeleton" version of the application (all key architectural components are in place and connected, but there is no "meat", so we have a walking skeleton).

Somewhere along the way you should reach the "order value" milestone - where you have 2 inputs ("# of items" & "price per item") and multiply them. No

discounts/states.



What is your next slice, after order value? (will probably be too big - challenge them to make it smaller!). Sample backlog:

Order value. 2 inputs, 1 output. No state tax, no discounts.
Hard-coded state tax. Still 2 inputs, 1 output. Utah state tax added automatically. (Now we can go live in Utah!).
3 inputs (price per item, # of items, state tax %). Output is order value with state tax added.

User inputs actual tax rate rather than state code. This gives simpler code (no internal data structure to map state code to tax rate), so we get faster delivery.

3 inputs (price per item, # of item, state tax), but only two states are supported. Any other gives error message.
Add the other 3 states. (discuss why there is limited value in adding 1 state at a time at this point - there is no risk reduction value, and it takes literally just a few seconds more to all 3 states at once)

Key point: Minimum key strokes per slice! We want maximum value with minimum effort (= Lean!)
5 minutes: make your slices smaller. Try for at least 15 slices.

How will you work? (optional)

This workshop is primarily about story slicing, but adding a tech practices aspect gives it extra spice.

How will you write & test your code? Make a decision & stick with it. Hold yourselves accountable! I reserve the right to harass any group that doesn't.

Option 1: TDD. By-the-book TDD. Red Green Refactor. Start each slice by writing a test that runs but fails. Then make the test green. Then refactor the code and make it clean.
Option 2: Red-green. Same as above, but refactoring is optional.
Option 3: Some tests. Will write some tests, but not for all slices, and not necessarily test-first.

Option 4: NFT (no f*ing tests).

Will you pair-program (= switch frequently), or have one business person & one programmer (business person tests, gives feedback, and improves the backlog)? Either is fine.

Build it

40 minutes, 5 iterations, 8 minutes per iteration.

At the end of each iteration, I will call out **"demo time!"**. That means stop coding, and demonstrate your app to another team. Then get back to coding.

Time doesn't stop between iterations! So don't spend too much time on demo.

Shout "slice" whenever you finish a slice.

Go!

(start 8 minute timer. Remember to restart it immediately after each sprint. Also keep careful track of which iteration we are on right now, it's easy to forget).

Review

How far did you get? (mark each team's approximate position on the value curve)

(typical result: some teams get past 5 states & 5 discounts. Most teams at least get to hard-coded state tax).

How many slices did you have?

Acceptance test:

Start your application and enter these values: I am in Utah, I'm buying 978 items, each item costs $270.99. (or pick whatever values you want).
Tell me the output! (write each team's output on the whiteboard).
No fiddling around, just run the app and enter the values and see what comes out.
(compare the results. Often the results differ, which is funny. Discuss possible reason for this. Some teams don't support decimal numbers for item price, discuss false assumptions & how to reveal them).

Debrief

Non-programmers: What was it like?

Programmers: What was it like?

How is your **code quality**, how proud are you of your code? (each programmer hold up 1-5 fingers).

TDD people should be 4 or 5, otherwise they have skipped or misunderstood the refactoring step in TDD.

Most programmers will be 1-2. Discuss how this is a potential pitfall of short iterations & small stories. Discuss importance of sustainable pace & quality, and how developers are responsible for this (Basic agile principle: *team chooses* how much work to pull in). Discuss “perceived” pressure vs real pressure.

Any **other questions or reflections**?

Round-robin: **What did you learn? What will you do?** Name one take-away insight from today, and one thing you will do differently in the future