## stories - what are they?

1. a promise to hold a conversation
2. the recorded details are never enough
3. a delta to the system
4. a place to document decisions

### notes

1. do we call victory when the feature (the solution) is implemented or when the benefit is delivered?
2. can we ever document everything that can exist around the story?
3. we have conversations because we know not everything is documented

## stories - a false user story

As a user

I want to be locked out of the system

When I get my password wrong 3 times

[David Evans]

### notes

This is a false **user** story. It's very easy to assign a user to a story without having asked that user.

## stories - forms

As a ”persona”

I want ”solution”

So that ”benefit”

### notes

1. it puts the user first
2. this form makes it easy to forget the ”so that...”

## stories - forms

In order to "benefit"

As a "persona"

I want "solution"

### notes

1. it puts the benefit yielded by the story first

## stories - forms

In order to "relative term" "benefit"

We will "verb" "solution"

### notes

1. forces the author to provide a relative term to the benefit rather than just a benefit. e.g. increase click-through rates
2. forces the solution to start with a verb e.g. allow multi-day parking in one call
3. can use the "verb solution" as what to write on the card "allow multi-day parking in one call"

[David Evans]

## Stories - Splitting - Why?

Are you splitting a story to fit into an iteration? This is splitting a story for an arbitrary reason. You can make your stories bigger or smaller by lengthening or shortening your iteration.

Here are some better reasons for splitting stories:

- A story can be split to make it less complex.
- A story can be split to deliver the most important features first.
- A story can be split to allow for emergent design.

# Stories - Splitting - Incremental Features - 1

## Setup

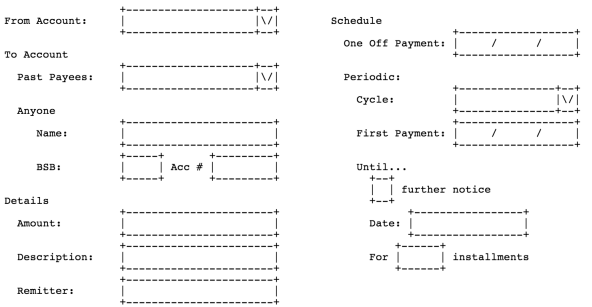


Figure 1: Draw on the whiteboard the following online banking site. This is the end state or what the customer wants.

# Stories - Splitting - Incremental Features - 2

## Setup

Enumerate the features on the board:

- pay anyone
- one off payment
- future payment
- recurring payment open ended
- recurring payment future dated
- recurring payment end dated
- pay previous payee

# Stories - Splitting - Incremental Features - 3

### the exercise

The group has to suggest how they would split the work and which piece of work they would do first.

### notes

- The one off payment to anyone is the easiest thing to do
- Redirect discussions of splitting the work based on "make the UI but have the buttons not work" back to a useful discussion about delivering incremental functionality.

## Stories - Splitting - Acceptance Criteria

Flesh out the acceptance criteria for a story, and this is based on 'separation of concerns' approach to splitting criteria.

Could the story go without one or two criterion? If so, could these be cleaved off into their own story?

# Stories - Splting - Technical Complexity - 1

This technique allows for both the fostering of new ideas as well as the identification of the first story with the least amount of effort. In our example we are going to plan for a 'Mobile Offline Wikipedia' application.

The required functionality is:

- download the data onto the device
- building an index for searching
- searching and search results
- formatting of a wiki page
- marking a page as a favourite

# Stories - Splting - Technical Complexity - 2

Draw on the whiteboard a five by five grid of boxes like so:

| no frills | | initial | | bells & whistles |
|---|---|---|---|---|
| | | download data | | |
| | | indexing | | |
| | | search & results | | |
| | | page display | | |
| | | favourites | | |

Muddy the waters by mentioning that downloading data needs to consider failed downloads, the 4GB limit on android operating systems, updates and slow connections. The search results need to handle no results found, one result and pagination. Result display needs to handle the plethora of device resolutions in existence.

# Stories - Splting - Technical Complexity - 3

Ask the group to suggest how they could do it differently. Put each answer in a box. Simpler solutions go on the no frills side and fancy features and exciting uses of technology go on the bells & whistles side.

**Example**

| no frills | | initial | | bells & whistles |
|---|---|---|---|---|
| copy from PC | basic download | ... | restart failures | auto-sync |
| alphabetic | subject index | ... | tailored index | |
| exact match | partial match | ... | autocomplete<br>fuzzy match | search suggestion<br>voice recognition |
| plain text | portrait only | ... | font size<br>custom styles | many resolutions |
| none | | ... | sync'd favs | |

# Stories - Splting - Technical Complexity - 4

You will see that most groups are very good at adding exciting new features on. Once they have filled up that side of the board suggest that their are simpler solutions out there if they think about it. The exercise isn't over until the left-most column is populated.

Once complete you can then show the team that the first story they should play is the left most column. This is the MVP that will provide all the functionality requested and prove the system. Once complete and feedback has been received the product owner can then cherry pick features off the grid as single stories. Autocomplete is a single story. As is partial matching.

A team will need to do this exercise for each feature they want to implement.

## knowledge transfer - business cards

1. get a stack of old business cards
2. host a session with the team to determine areas of the product
3. write each area on a business card
4. put all the business cards in a box
5. give the box to a new hire
6. the new hire has 2-3 weeks to use each card
7. the new hire can take any card from the box and ask anyone in the team about the area
8. if that person doesn't know then they have to come with the new hire to find someone that does

[Per Hallstrom]

# Acceptance Criteria Games - 99 Balloons - 1

## equipment

- a bag of balloons (all the same shape)
- 20c and 10c coins (or equivalent)
- string & ruler, or a flexible ruler

## before the session

1. before the session: blow up a balloon and using the coins to draw two eyes and a nose. Draw a smiling face and optional eyebrows. Make a note of the dimensions of the balloon (perimeter, height, distance between eyes, etc).

# Acceptance Criteria Games - 99 Balloons - 2

## starting the session

1. split up the class into teams of about 4
2. give each team a mound of balloons (keep the coins, string and ruler hidden)
3. tell the groups that they need to make me 99 balloons; hold up your pre made balloon and tell them that they need to make theirs like this one. They have 2 minutes. Go!

## during an iteration

1. don't help
2. keep track of time

# Acceptance Criteria Games - 99 Balloons - 3

## ending an iteration

1. examine each balloon compared to the original; discard it if it is even slightly different to original
2. be visibly disappointed in the lack of accurate balloons
3. ask the groups is "making this balloon" was not simple enough?
4. don't tell them any answers, it is there responsibility to ask questions to clarify the intent
5. answer all yes/no questions "should it have two eyes" with a yes/no. Only elaborate when they ask leading questions "should the eyes be of a certain size?"

# Acceptance Criteria Games - 99 Balloons - 4

## length of session

1. about 3-4 iterations should be required
2. keep the score on the whiteboard of each team and how many good balloons and how many 'reject' balloons have been made.
3. the group should become frustrated with the constant rejections and will eventually start to ask questions; this is the point

# Acceptance Criteria Games - 99 Balloons - 5

## summary

1. have a discussion about asking clarifying questions and the difference between closed (answer is chosen from a set e.g. yes/no) and open (tell me about...) questions
2. also focus on the questions that were asked that confirm what the group already knows vs. uncovers something not yet known by the group.
3. give kudos to anyone asks "why" we are making balloons. An answer is to make children happy (which can lead into a discussion about whether balloons will do that and user feedback)

## Acceptance Criteria - The Good

- don't document things outside of your domain (i.e. how to parse a CSV file)

# testing - confirming questions - 1

## the exercise

1. Write on a white board the following sequence: 2,4,6,8
2. Tell the group that they can suggest any number, you will say if it is in the sequence or not. Write each number they say at the end of the sequence, if it is in the sequence.
3. Write numbers that are not in the sequence elsewhere.
4. When the group thinks they have enough information then they should yell out the sequence

# testing - confirming questions - 2

## notes

- The actual sequence is that each number must be greater than the prior number.
- Many groups will continue the sequence 10,12, etc
- If they answer anything other than the correct sequence; talk to the group about the types of questions they asked. Numbers like 10, 12, 14, etc just confirm what the group already knows. Numbers like -50, 5, 12.5 and 1 billion all challenge what is currently known.

## testing - automation - spectrum - 1

All testing sits upon a spectrum between manual and automatic. There is value all along the spectrum.

## testing - automation - spectrum - 2

The manual test is exploratory. What is covered changes with each execution and changes during execution as the tester responds to and learns from the behaviour of the system.

The semi-automatic test is each little piece of code that helps you undertake a test, but doesn't do the test for you. A script that prepares an input file from a spreadsheet, a button you can press to lodge an order that would otherwise take 20 minutes across several systems.

The automatic test includes all the unit tests out there. These run frequently, often without direct user input (i.e. as part of a continuous integration server).

## performance - risk - identify

- valuable (economic) system output - especially for outliers
- tight (realistic) time oriented requirements
- data volumes high (number of objects - not just total, but distribution across customer / products - maybe outliers matter)
- transaction volumes high
- data sizes large (the width of the object)

If we have two or more than we have a risk.

## performance - risk - mitigate

- relax the requirement
- data review
- spike solution
- design review
- code review
- explotory testing
- performance testing