Agile Software Development

Lecture 4: Gathering Requirements Knowing what the customer wants?

Mahmoud El-Gayyar

elgayyar@ci.suez.edu.eg

Slides are a modified version of the slides by Prof. Kenneth M. Anderson

Outline (Gathering Requirments)

- *Requirements and Requirements Gathering*
  - Brainstorming
  - User Stories
  - Planning
    - *Estimation Game*

Requirements Gathering

- *Requirements gathering begins with a problem statement*

We need a web site showing our current deals, and we want our users to be able to book shuttles and special packages, as well as pay for their bookings online. We also want to offer a luxury service that includes travel to and from the touristic places and accommodation in a local hotel

- *Characteristics?*
  - informal, unstructured, all over the place (deals, bookings, packages, payment, shuttle services, hotels)

First Step: Impose Structure

- *Identify all of the different things the system has to do.*

Title: **Description:**	**Title:** **Description:**
Title: **Description:**	**Title:** **Description:**

In particular: Find Requirements

- *A requirement is a **single thing** that the software has to do*

Title: Show Current Deals

Description: The website will show current deals to Orion's Orbits users.

✓ Written in ***User's Language***

✓ ***Informal:*** because we don't have a lot of information But, allows us to validate initial understanding of domain

Translate Entire Problem Statement

Title: Show Current Deals

Description: The website will show current deals to Orion's Orbits users.

Title: Book a shuttle

Description: An Orion's Orbits user will be able to book a shuttle between hotel and touristic place.

Title: Book package

Description: An Orion's Orbits user will be able to book a special package with extras online.

Title: Pay online

Description: An Orion's Orbits user will be able to pay for their bookings online.

Title: Arrange Travel

Description: An Orion's Orbits user will be able to arrange travel to and from the hotel.

Title: Book a hotel

Description: An Orion's Orbits user will be able to book a hotel.

Then, return to customer and

- *ask questions*
  - Did I get this right?
  - What did you mean by…
- *and gather more requirements*
  - Is this really all of the functionality that you need?
  - If we built all of this, what would you want in version 2.0?

All this work will lead to new or clarified requirements

Problem: Not Enough?

- *One problem that you'll encounter is that this back and forth may not be enough to get to crisp detailed requirements*
  - or you feel that you just don't have a good grasp on the big picture
- *This can be especially true if "customer" ≠ "end user"*
- *Next step is to hold a **brainstorming** session with as many different stakeholders as possible*
- *what the book calls a "**bluesky session**"*

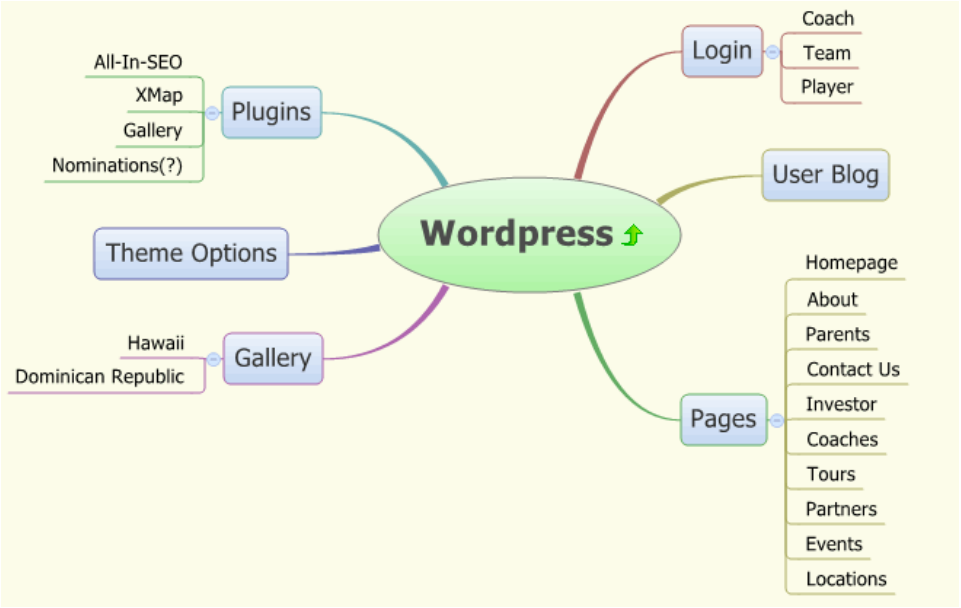
Bluesky Session

- *Brainstorming session*
  - **Goal:** get stakeholders to generate tons of candidate requirements; not everything will make it into the final system

- *Things to Avoid*
  - **The Silent Tomb®**: Leave job titles at the door, people should not feel afraid to speak up just because the boss is there
  - **Criticizing people** rather than ideas
  - **Developer jargon** "NOT 'AJAX' but 'rich user interface'"

Tool Support (CASE)

- *Make use of outliners and other types of note taking applications during brainstorming sessions to capture the generated ideas, domain knowledge, and requirements;*
- *XMind (free) (Brainstorming and mind mapping tools)*



Gray Skies

- *If things go wrong during the bluesky session: "bad boss"*
- *Make use of other techniques*
  - **Interview** end users and have them pretend to interact with their "ideal system", what the book calls "**role playing**"
  - **Observe** them working on tasks related to the system
    - *how would the task change if the system were present?*
  - Review the documents they use now
    - *ask if the document would go away if the system were present or how would it change?*

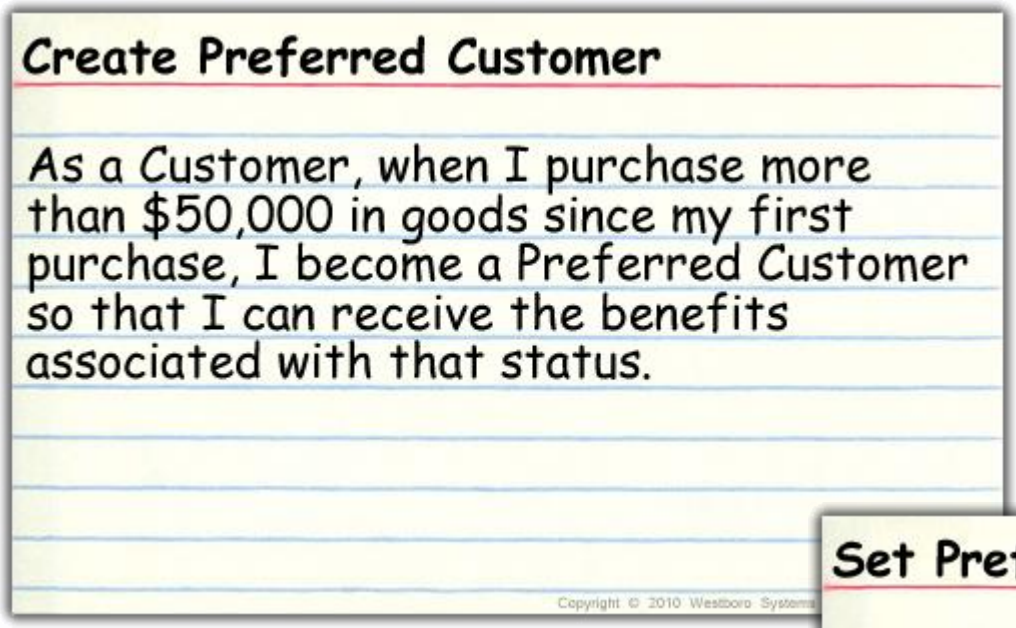
Next? User Stories

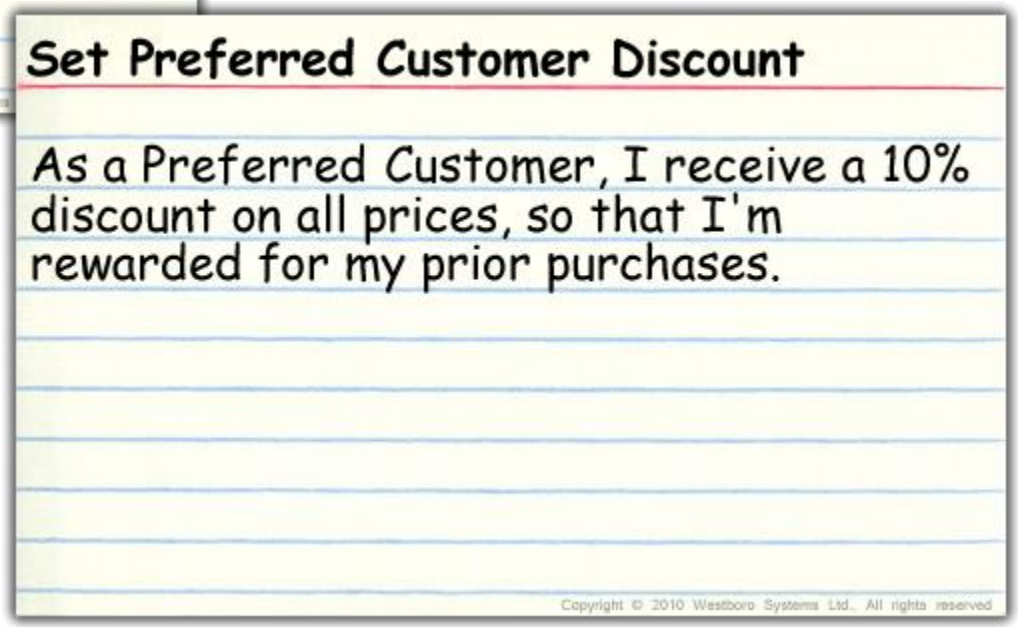
- *Transform requirements gathered so far into user stories*
  - A user story describes how the user interacts with the software you're building
  - It should be written from your customer's perspective and describe what the software is going to do for the customer
- *User stories are essentially **informal use cases***

User Stories

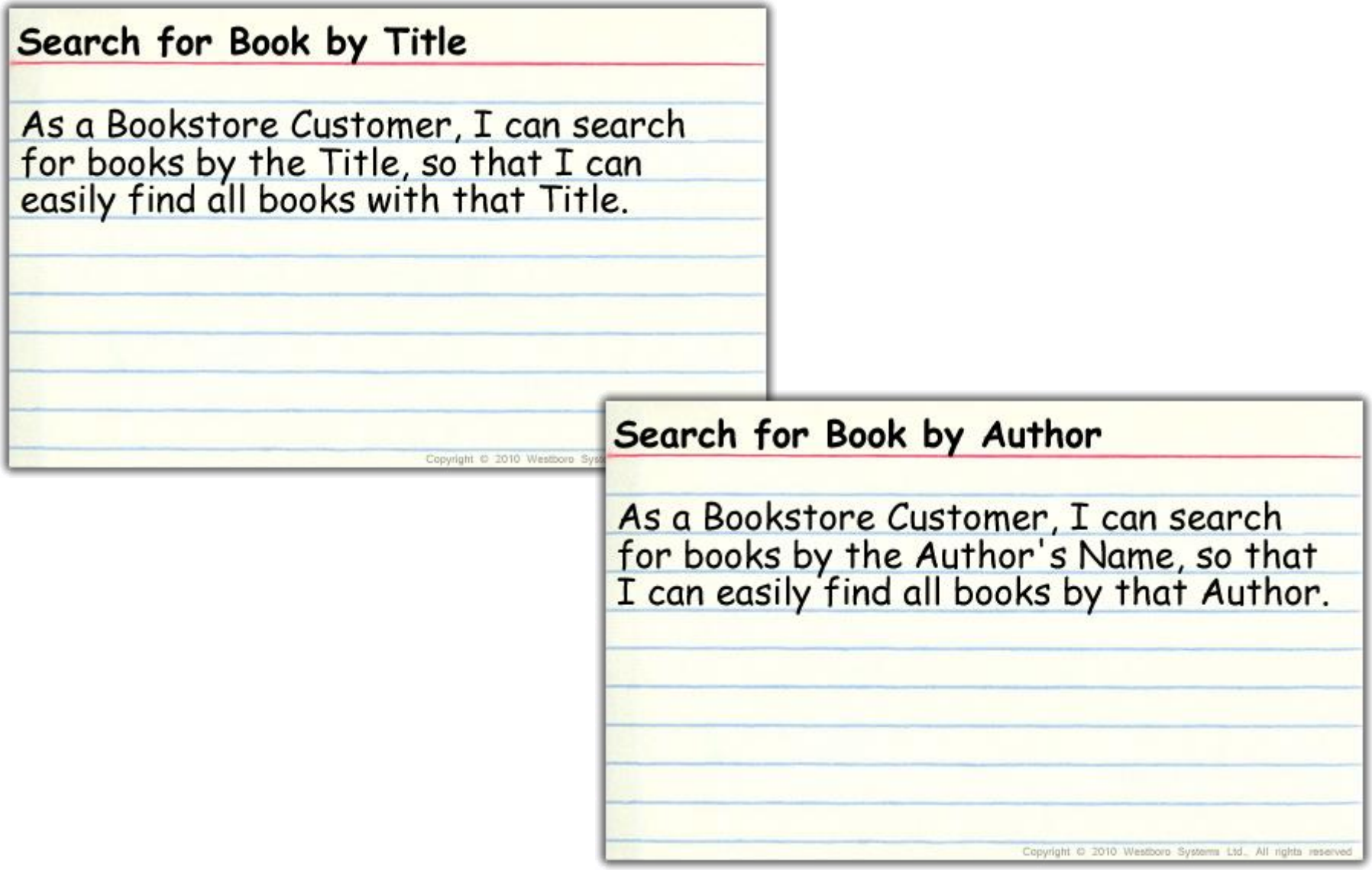
- *SHOULD*
  - describe ***one thing*** the system should do for the customer
  - be written using ***language*** that the ***customer*** understands
  - be written by the ***customer*** !!
  - ***be short***. No longer than three sentences
- *SHOULD NOT*
  - be a long essay
  - use technical terms unfamiliar to the customer
  - mention specific technologies (save those for design)

Example 1: User Story



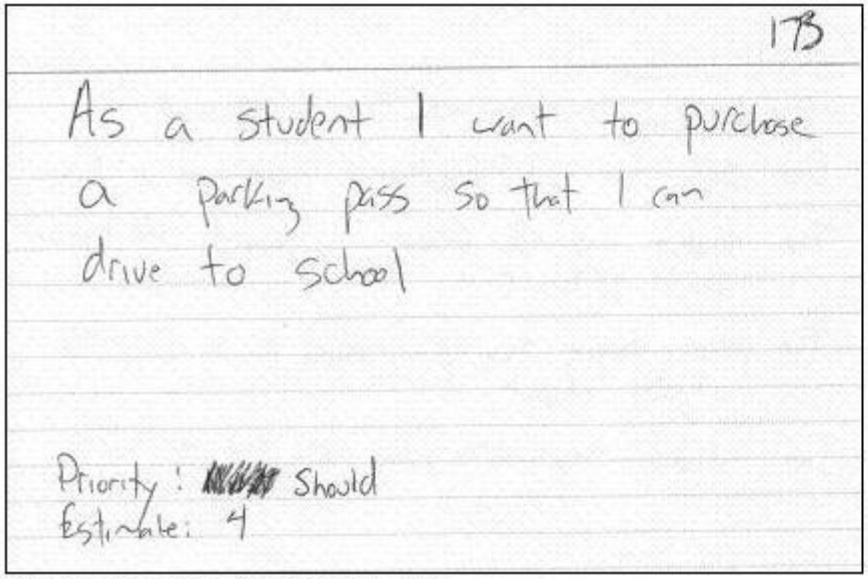


Example 2: User Story

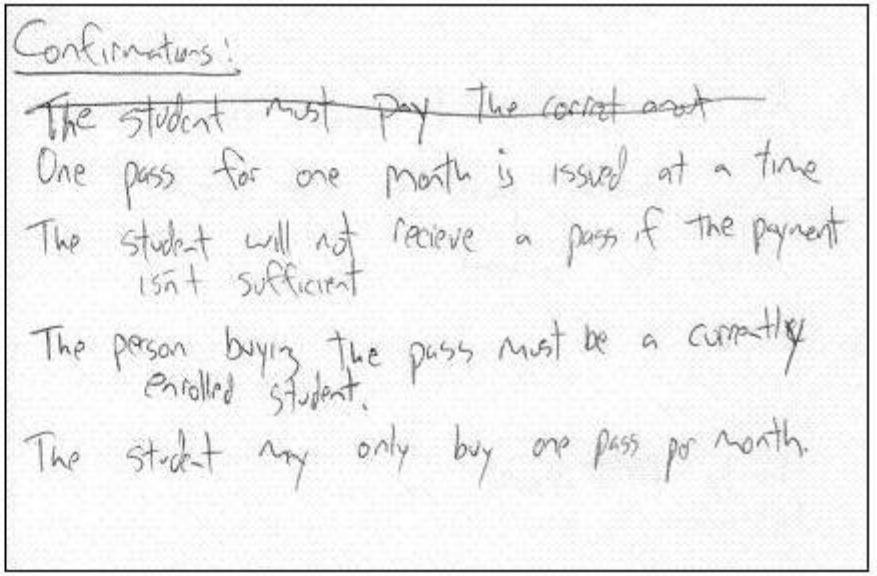


Example 3: User Story

Front of Card



Back of Card



Requirements Life Cycle

- *We now have a life cycle for use at the start of a project*
  - Capture basic ideas from problem statement
  - Return with first pass, ask questions, set-up bluesky session
  - ***ITERATE***
    - *Construct User Stories*
    - *Find holes with stories and fix them with customer feedback, find new requirements, ask questions to assess completeness*
  - Finish with initial set of clear, customer-focused user stories
- *This defines the **WHAT** of the project, next up is the **WHEN***

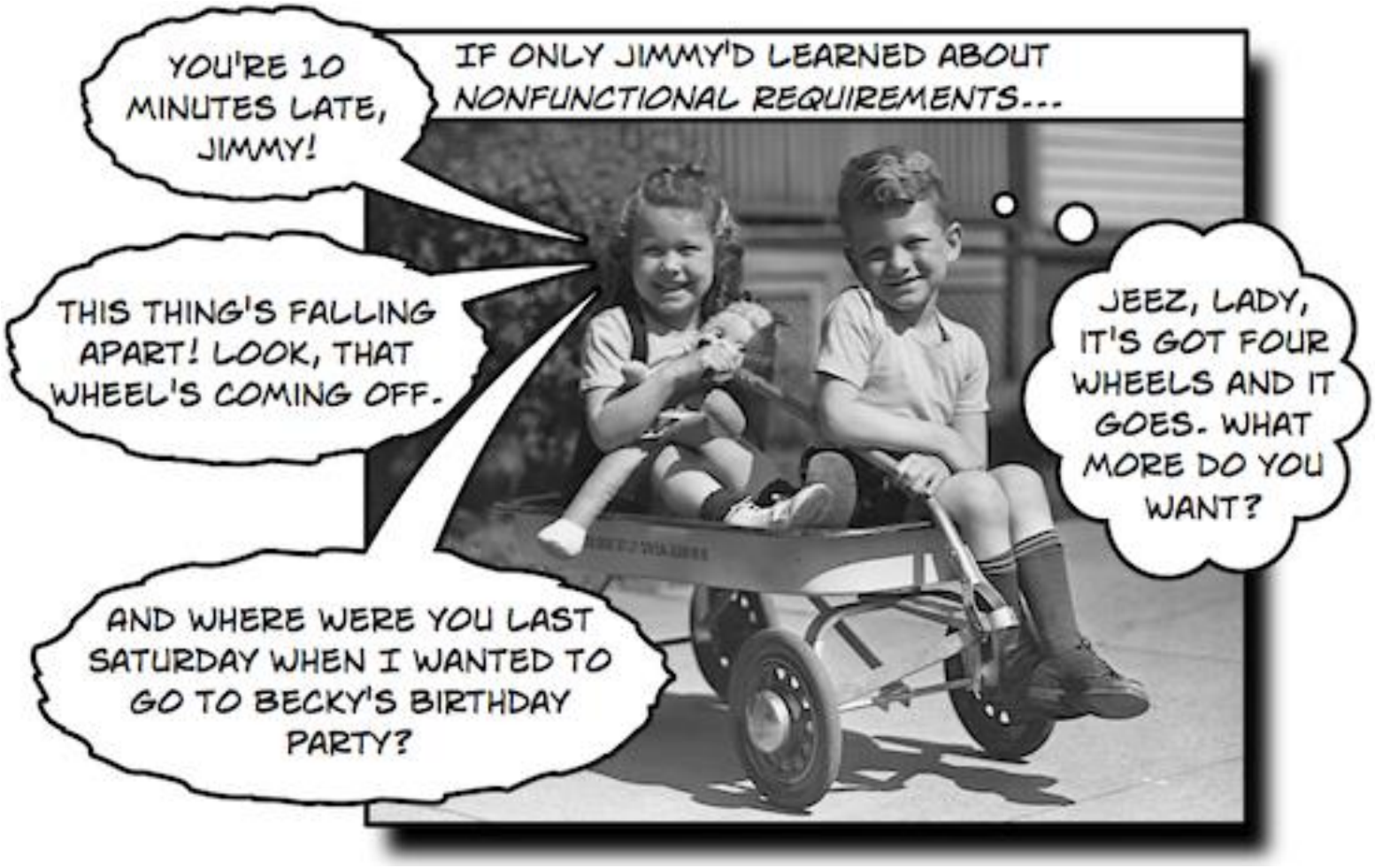
But First… Types of Requirements

- *There are different types of requirements*
  - functional
  - non-functional
  - constraints
- *The process that we described above is focused on generating **functional requirements***
  - what are the functional capabilities of the proposed system

Non-functional Requirements 1

- *A non-functional requirement*
  - states things that are true of a system regardless of its functionality
- *For instance:*
  - The system will return a response in less than a second
  - This is a non-functional requirement on performance: note that it provides no conditionals; for whatever reason, it wants sub-second response time no matter how many users the system it has or how much data it is processing

Non-functional Requirements 2



Non-functional Requirements 3

- *Non-functional requirements are sometimes called the "ilities"*
  - reliability
  - extensibility
  - flexibility
  - scalability (in terms of users, data, machines, etc.)
- *but also*
  - robustness, security, fault tolerance, performance
- *Main point: these are requirements independent of the system's core functionality*

Constraints

- *Constraints in the requirements phase are typically restrictions imposed by your client on the range of possible solutions to the stated problem*
  - "We're a Windows shop; the system has to run on XP"
  - "We've already bought Oracle; you'll need to use it"
- *You want to avoid these as much as possible*
  - it can limit your creativity as a designer
  - but often you'll have at least one or two in any given project

(On to) Estimates

- *At some point during the requirements gathering process, the customer will ask*
  - How long will all of this take to develop?
- *You need to supply a project estimate*
  - which will be the sum of the estimates for your user stories
- *So, now you need to supply estimates for each user story*
  - How do we come up with this estimate?

Planning Poker - 1

- *A popular estimation technique in agile methods*

Planning Poker - 2

- *Addresses the problem in which two or more team members come up with wildly different estimates for a story*
  - i.e. when a single user story generates estimates of say “3 days”, “2 weeks”, and “3 months” from three different developers
- *The underlying cause for these different estimates is assumptions; what did you assume was true or not true about the project to generate the number that you did?*

Example: Different Estimates 1

- ***Task:*** *"Add a comment on a product page"*
- *One team member might think:*
  - "Simple. We need a form, a script to process the form, and a place to store the comment in the database. 3 days."
- *Another might think:*
  - "Hmm. How do we relate the comment to the product? Do we have one comment table per product in the database? Will I need to change the product class? Maybe there is code from some other place in the system that I can re-use. 2 weeks."

Example: Different Estimates 2

- ***Task:*** *"Add a comment on a product page"*
- *Finally, another might think:*
  - "Ugh. Complete database re-design. No code to re-use (this is the first time we're allowing comments). What user interface should we use? Can the user embed HTML in their comments? How do we handle smileys? How will this impact the product model class? Do we keep the comments forever? Do we need moderation? Can a user edit a past comment? Who gets to delete comments? Yuck!! 3 months!"
- *Based on your assumptions, you'll get completely different numbers. How do you get these assumptions to the surface?*

  Planning Poker!

Planning Poker I

- *Create "deck" of cards. 13 cards per "player".*
  - Each card contains an estimate spanning from "already done" to "wow this is going to take a long time".
  - 0, .5, 1, 2, 3, 5, 8, 13, 20, 40, 100 days
  - One card has a "?" meaning "not enough information"
  - One card has a coffee cup meaning "lets take a break"

Planning Poker II

- *Place a user story in the middle of the table*
  - Each team member thinks about the story and forms initial estimate in their heads
  - Each person places the corresponding card face down on the table; note: estimate is for entire user story
  - Everyone then turns over the cards at the same time

Planning Poker III

- *The larger the difference between the estimates, the less confident you are in the estimate, and the more assumptions you need to highlight and discuss.*
- *So, the next step in planning poker is*
  - Put assumptions on trial for their lives
  - Have each team member list the assumptions they made and then start discussing them
  - Again, you need to criticize the assumption not the person
- *Goal is to get agreement on what assumptions truly apply*

Planning Poker IV

- *If the assumptions reveal a misunderstanding of the requirements, then go back to the client and get that misunderstanding clarified*
- *Otherwise, start to eliminate as many assumptions as possible, then have everyone revise their estimates and play planning poker again to see if the spread has decreased*
- *Your goal is convergence. Once estimates cluster around a common number, assign that number and move to the next story*

Planning Poker V

- *Your life cycle is thus*
  - Talk to customer: clarify misunderstandings, assumptions
  - Play planning poker
  - Clarify assumptions, possibly by returning to step 1
  - Come to a consensus estimate for the user story
- *Do this until all user stories have a consensus estimate assigned*

Planning Poker VI

- *Things to watch out for*
  - Big estimates (== bad estimates)
    - *They indicate that the story is too big; decompose; try again*
    - *Remember, the book's ideal iteration is 20 work days (1 month)*
    - *Estimates longer than 15 days are more likely to be wrong than those shorter than 15 days; (others think 7 days is upper limit)*

What to do – Big estimates ?

- *Break your story into smaller ones (Using the AND rule)*
  - Any user story that has an AND in its title or description can probably be split into two or more smaller ones
- *Talk to your customer again*
  - May be there are some assumptions that push your estimation up. By talking with your customer, you may get a better understanding and those assumptions might go away.

Your time !!

- *Exercise 3 : Break down stories*

Requirements Life Cycle

- *Capture basic ideas*
- *Bluesky Brainstorming*
- *Construct User Stories*
- *Find holes, get feedback*
- *Clear, customer-focused user stories*
- *Play planning poker*
- *Clarify misunderstandings and assumptions*
- *Develop project estimate*

Key Points

- *In this lecture, we focus on the importance of requirements (functional, non-, constraints)*
- *Of identifying the problem to be solved*
- *Of the use of iteration in requirements gathering*
- *How user stories can be used to document requirements*
- *How planning poker can be used to generate estimates*