Product Backlog

Lecturer: Ngo Huy Bien
Software Engineering Department
Faculty of Information Technology
VNUHCM - University of Science
Ho Chi Minh City, Vietnam
nhbien@fit.hcmus.edu.vn

Objectives

- To create a *product backlog*.

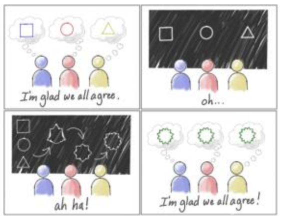
Contents

References

1. Jeff Patton and Peter Economy (2014). User Story Mapping. O'Reilly Media.
2. Kenneth S. Rubin (2012). Essential Scrum A Practical Guide to the Most Popular Agile Process. Addison-Wesley Professional.
3. Jonathan Rasmusson (2010). The Agile Samurai - How Agile Masters Deliver Great Software. Pragmatic Bookshelf.
4. Mike Cohn (2010). Succeeding with Agile - Software Development Using Scrum. Addison Wesley.
5. Mike Cohn (2004). User Stories Applied For Agile Software Development. Addison-Wesley.

Why Stories? [1]

- The idea is that if I have an idea in my head and I describe it in writing, when you *read* that document, you might quite possibly imagine *something different*.

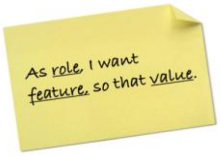


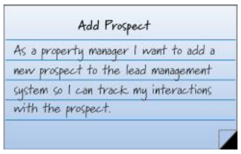
Therefore

- Stop trying to write the *perfect* document.
- Go ahead and *write something*, anything. Then use *productive conversations* with words and pictures to build shared understanding.
- The real goal of using stories is *shared understanding*.

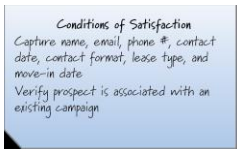
User Stories [2]

- *User stories* are a convenient format for expressing the desired business value for many types of product backlog items, especially features.


As role, I want feature, so that value.


Add Prospect

As a property manager I want to add a new prospect to the lead management system so I can track my interactions with the prospect.

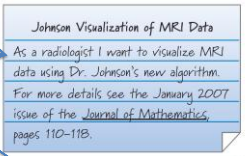

Conditions of Satisfaction

Capture name, email, phone #, contact date, contact format, lease type, and move-in date

Verify prospect is associated with an existing campaign

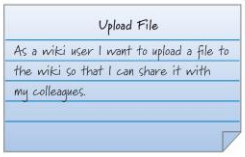
The Three Cs

- Card, conversation, and confirmation.

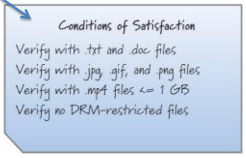

Johnson Visualization of MRI Data

As a radiologist I want to visualize MRI data using Dr. Johnson's new algorithm.

For more details see the January 2007 issue of the Journal of Mathematics, pages 110–118.


Upload File

As a wiki user I want to upload a file to the wiki so that I can share it with my colleagues.


Conditions of Satisfaction

Verify with .txt and .doc files

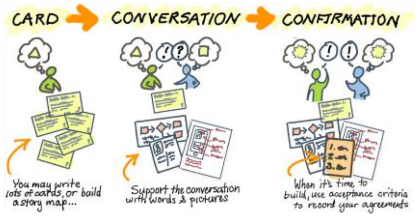
Verify with .jpg, .gif, and .png files

Verify with .mp4 files <= 1 GB

Verify no DRM-restricted files

Again [1]

- Story *conversations* are about working together to arrive at a best solution to a problem we both understand.


CARD
CONVERSATION
CONFIRMATION
You may write lots of cards, or build a story map...
Support the conversation with words & pictures
When it's time to build, use acceptance criteria to record your agreements

Words and Pictures

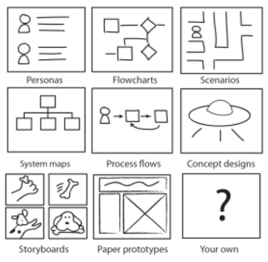
Gathering Stories [2]

- Traditional approaches to requirements gathering involve asking the users *what they want*.
- I have *never* been very successful with that approach.
- In my experience, users are far *better critics* than they are authors.
- A better approach is to involve the users as part of the team that is *determining what to build* and is *constantly reviewing* what is being built.
- To promote this level of participation, many organizations prefer to employ *user-story-writing workshops* as a principal means of generating at least the initial set of user stories.

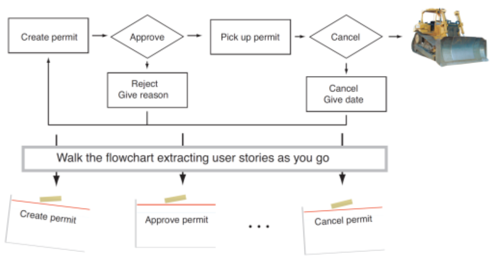
User-Story-Writing Workshop

- If it is the *first workshop*, I usually start by performing user role analysis.
- The goal is to determine the *collection of user roles* that can be used to populate the user role part of our stories **("As a <user role>, I want to . . .")**.
- Of course, *marketing* or *market research people* might have created a good definition of our users in a separate activity prior to the story-writing workshop.
- We might also have *personas*, which are prototypical individuals that represent core characteristics of a role.

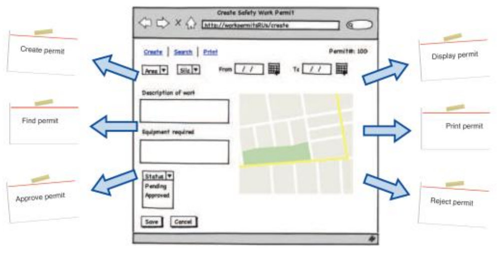
Draw Lots of Pictures [3]



Use Flowchart



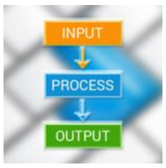
Breakdown A Screen



Brainstorm Everything Else

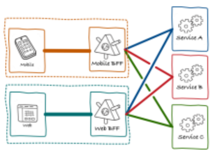
Issue 1 [4]

- "I can't possibly put my requirements on index cards."
- Do you have any inputs (*business problems*, *business goals*, *process flow*, *UI flow*)?
- Did you try using pen-and-paper instead of a tool?



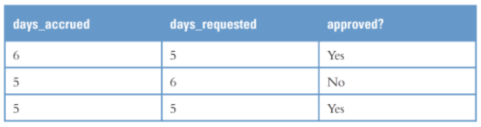
Issue 2

- "We write *back-end software* that no users ever see, so user stories don't make sense for us."
- A story that reads, "*As the loan authorization system, I want to receive all data as valid, well-formed XML so that I don't have to worry about syntax checking," is perfectly valid.*



Specify By Example

- *"As an employee, I want a request for up to my earned vacation time to be automatically approved so that I don't need to wait for someone to approve it manually."*
- Examples showing that a request for more time off than has been accrued will not be automatically approved.

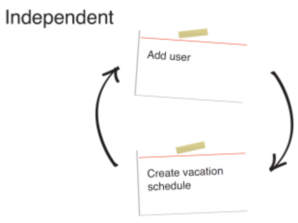


days_accrued	days_requested	approved?
6	5	Yes
5	6	No
5	5	Yes

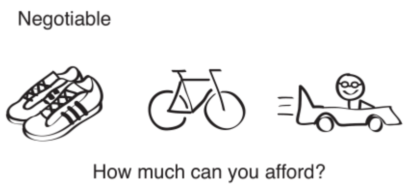
INVEST in Good Stories [2]

- I – Independent
- N – Negotiable
- V – Valuable
- E – Estimable
- S – Small (+ Screens)
- T – Testable

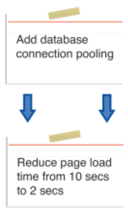
Independent [3]



Negotiable



Valuable


Add database connection pooling
Reduce page load time from 10 secs to 2 secs

So, whenever you're tempted to write something technical as a story ...

... instead try to rewrite it in terms of something valuable to the business.

Something they can get excited about!

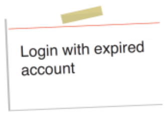
Estimable & Small

Small and Estimatable

Think you boys can get this done in a week?

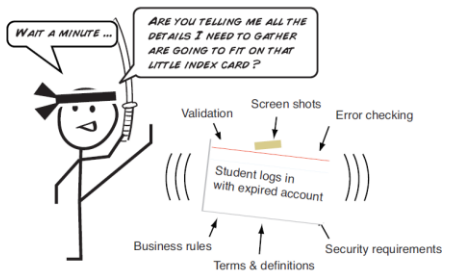
Testable

Testable

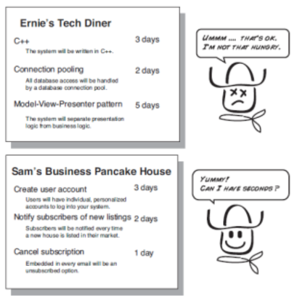


- Allow regular logins
- Re-direct expired logins
- Display appropriate error message
- Handle nonexistent user account

Index Cards Remind Us *NOT* to Try to Write Everything Down


Wait a minute ...
Are you telling me all the details I need to gather are going to fit on that little index card?
Validation
Screen shots
Error checking
Student logs in with expired account
Business rules
Terms & definitions
Security requirements

Which Restaurant Would Your Hungry Customer Rather Dine At?


Ummm .... that's ok. I'm not that hungry.
Yummy! Can I have seconds?

Ernie's Tech Diner

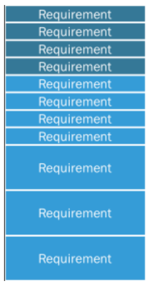
C++	3 days
The system will be written in C++.	
Connection pooling	2 days
All database access will be handled by a database connection pool.	
Model-View-Presenter pattern	5 days
The system will separate presentation logic from business logic.	

Sam's Business Pancake House

Create user account	3 days
Users will have individual, personalized accounts to log into your system.	
Notify subscribers of new listings	2 days
Subscribers will be notified every time a new house is listed in their market.	
Cancel subscription	1 day
Embedded in every email will be an unsubscribed option.	

The Product Backlog

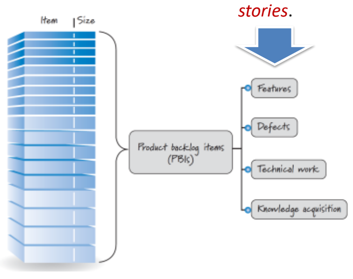
- The *product backlog* is a prioritized list of desired product functionality.
- It provides a centralized and shared understanding of *what* to build and *the order* in which to build it.



Backlog Items

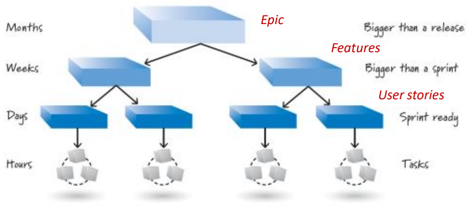
- The product backlog is composed of *backlog items*.
- Most PBIs are *features*, items of functionality that will have tangible value to the user or customer.

These are often written as *user stories*.

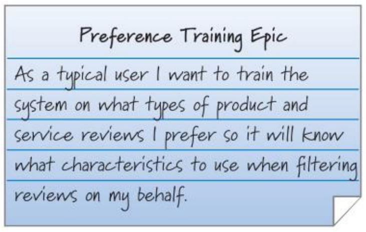


Level of Detail [2]

- If there is only one (small) size of story, we will be obligated to define all requirements at a *very fine-grained level* of detail long before we should.



Example Epic

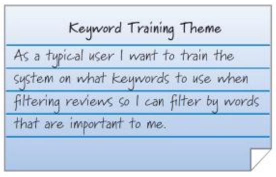


Preference Training Epic

As a typical user I want to train the system on what types of product and service reviews I prefer so it will know what characteristics to use when filtering reviews on my behalf.

Themes

- Some teams also use the term *theme* to refer to a collection of related stories.
- Themes provide a convenient way to say that a bunch of stories have something in common, such as being in the same functional area.



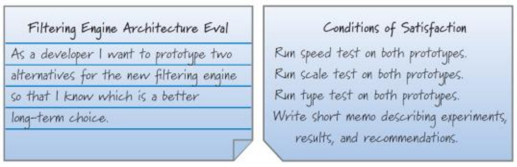
Keyword Training Theme

As a typical user I want to train the system on what keywords to use when filtering reviews so I can filter by words that are important to me.

Knowledge-Acquisition

- Sometimes we need to create a product backlog item that focuses on *knowledge acquisition*.
- Perhaps we *don't have* enough exploitable knowledge about the product or the process of building the product to move forward.
- Such exploration is known by many names: *prototype*, *proof of concept*, experiment, study, spike, and so on.
- They are all basically exploration activities that involve *buying information*.

Knowledge-Acquisition Example



Filtering Engine Architecture Eval

As a developer I want to prototype two alternatives for the new filtering engine so that I know which is a better long-term choice.

Conditions of Satisfaction

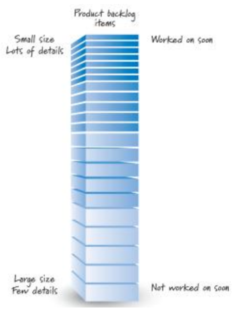
Run speed test on both prototypes.
Run scale test on both prototypes.
Run type test on both prototypes.
Write short memo describing experiments, results, and recommendations.

Knowledge-Acquisition Story Evaluation

- First, we need to know the *cost* of the prototyping.
- The team might <u>not</u> be able to answer particular questions until an architectural decision has been made, but it must be able to answer the question of *how much effort* it wants to spend <u>to buy the information</u> necessary to make the architectural decision.
- Now we need to know the value of the information. I ask the team to estimate the cost of *being wrong*.

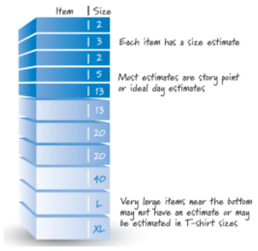
Detailed Appropriately

- Product backlog items are different sizes.

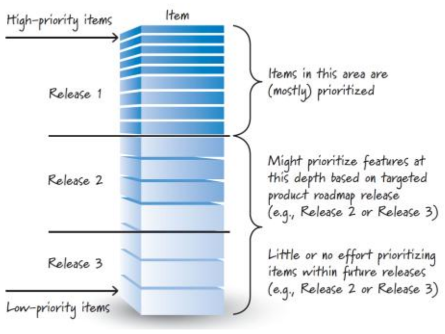


Estimated

- Each product backlog item has a size estimate corresponding to the effort required to develop the item

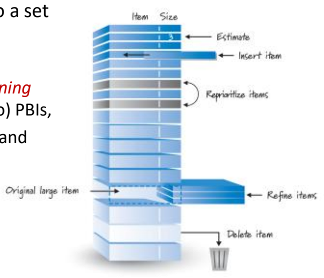

Item
Size
2
3
2
5
13
13
20
20
40
L
XL
Each item has a size estimate
Most estimates are story point or ideal day estimates
Very large items near the bottom may not have an estimate or may be estimated in T-shirt sizes

Prioritized


High-priority items
Item
Release 1
Release 2
Release 3
Low-priority items
Items in this area are (mostly) prioritized
Might prioritize features at this depth based on targeted product roadmap release (e.g., Release 2 or Release 3)
Little or no effort prioritizing items within future releases (e.g., Release 2 or Release 3)

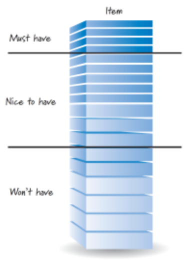
Grooming

- *Grooming* refers to a set of three principal activities:
  - creating and *refining* (adding details to) PBIs,
  - estimating PBIs, and
  - *prioritizing* PBIs.



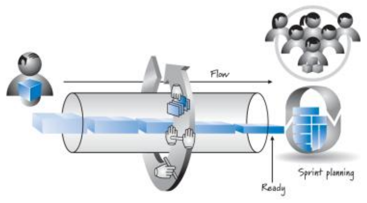
Release Flow Management

- The product backlog must be groomed in a way that *supports ongoing release planning* (the flow of features within a release).



Sprint Flow Management

- If the product backlog has been detailed appropriately, the items at the top of the backlog should be *clearly described and testable*.



Definition Of Ready

- *Business value* is clearly articulated.
- Details are sufficiently understood by the *development team* so it can make an informed decision as to whether it can complete the PBI.
- *Dependencies* are identified and no external dependencies would block the PBI from being completed.
- *Team* is staffed appropriately to complete the PBI.
- The PBI is *estimated* and small enough to comfortably be completed in one sprint.
- Acceptance criteria are clear and *testable*.
- *Performance criteria*, if any, are defined and testable.
- Scrum team understands how to *demonstrate* the PBI at the sprint review.

Scrum vs. Traditional Requirements

- If we're running out of time or money, we can *drop* low-value requirements.
- If, during development, new information indicates that the cost/benefit ratio of a requirement has become significantly less favorable, we can choose to *drop* the requirement from the product.
- If a new high-value requirement emerges, we have the ability to *add* it to the product, perhaps *discarding* a lower-value requirement to make room.

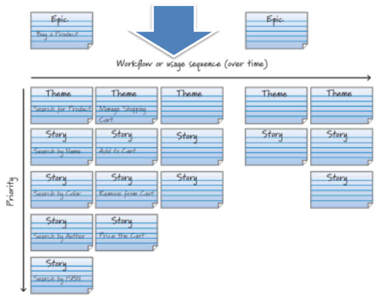
Traps [1]

- Because stories let you focus on building small things, it's easy to *lose sight of the big picture*.
- When you're building a product of any significant size, building one *small thing after another* leaves people wondering when you'll ever be done, or what exactly you'll deliver.
- Because stories are about conversations, people use that idea to *avoid writing anything down*. Then they *forget* what they talked about and agreed to in the conversations.

Story Mapping Technique [2]

- *Story mapping* is a technique popularized by Jeff Patton (Patton 2009) that takes a user-centric perspective for generating a set of user stories.
- The basic idea is to *decompose* high-level user activity into a workflow that can be further *decomposed* into a set of detailed tasks.
- Patton uses terms like *activity*, *task*, and *subtask* to describe the hierarchy inside a story map.
- To be consistent with the terminology I introduced earlier, I use *epic*, *theme*, and *sprintable story*.

Story Map Example

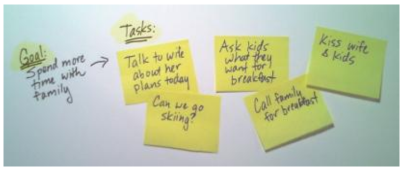


User Tasks

- Write out *your story* a step at a time
- User tasks are the basic *building blocks* of a story map.

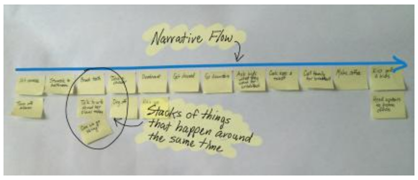
Goal-Level Concept

- Use the *goal-level concept* to help you aggregate small tasks or decompose large tasks.

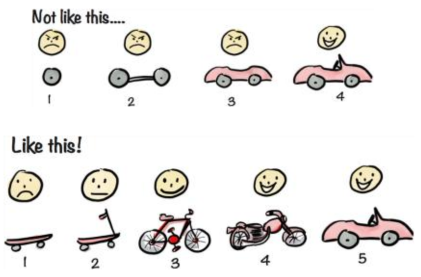

Goal: Spend more time with family
Tasks:
Talk to wife about her plans today
Ask kids what they want for breakfast
Kiss wife & kids
Can we go skiing?
Call family for breakfast

Storytelling

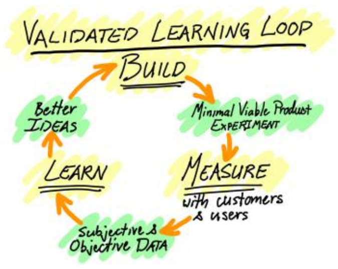
- Maps are organized left-to-right using a *narrative flow*: the order in which you'd tell the story.



Minimum Viable Product Experiment [1]



Product Discovery



Story Map vs. Product Backlog [2]

- *Story mapping* combines the concepts of <u>user-centered design</u> with <u>story decomposition</u>.
- *Good story maps* show a <u>flow of activities</u> from the users' perspective and provide a context for understanding individual stories and their relationship to larger units of customer value.
- Story maps provide a *two-dimensional view* of a product backlog instead of the traditional linear (one-dimensional) product backlog representation.

User Stories aren't Scenarios [5]

Maria is thinking about making a career change. Since the glory days of the dot-com boom she has worked as a tester at Big- TechCo. A former high school math teacher, Maria decides she'll be happier if she returns to teaching. Maria goes to the BigMoney-Jobs.com website. She creates a new account with a user name and password. She then creates her resume. She wants to find a job as a math teacher anywhere in Idaho but preferably near her current job in Coeur d'Alene. Maria finds a handful of jobs that match her search criteria. The job that intrigues her most is with the North Shore School, a private high school in Boise. Maria has a friend, Jessica, in Boise whom she hopes may know someone at North Shore. Maria enters Jessica's email address and forwards the job link to her with a note asking if she knows anyone at the school. The next morning Maria gets an email from Jessica saying that she doesn't know anyone at the school, but she knows of the North Shore School and it has a wonderful reputation. Maria clicks on a button that submits her resume to North Shore.

User Stories aren't IEEE 830

4.6) The system shall allow a company to pay for a job posting with a credit card.

- 4.6.1) The system shall accept Visa, MasterCard and American Express cards.
- 4.6.2) The system shall charge the credit card before the job posting is placed on the site.
- 4.6.3) The system shall give the user a unique confirmation number.

Just because something is boring to read is *not sufficient reason* to <u>abandon</u> it as a technique.

User Stories aren't Use Cases

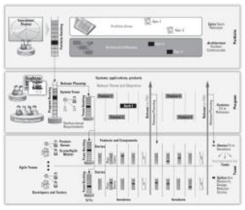
- The scope of a use case brief is usually larger than the scope of a user story.
- *Use case briefs* are intended to live on for the life of a product. *User stories*, on the other hand, are disposed of.
- *Use cases* are generally written as the result of an analysis activity, while *user stories* are written as notes that can be used to initiate analysis conversations.

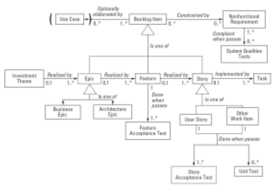
Question [4]

- "I'm already good with *use cases*; do I really need to switch to user stories?"
- Scrum teams do best with units of work that are *smaller than* a typical use case.
- So, although *you can have use cases* on your product backlog, be aware that you'll probably want to write far smaller ones than were intended by their originator.

Further Reading

- Dean Leffingwell (2011). Agile Software Requirements. Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison-Wesley Professional.
  - Agile requirements in the enterprise.





Thank You & See You Again