Lecture Notes
September 21, 2017

Pair Needs Team - Python or Java

2nd team assignment (project proposal) due next Tuesday September 26

3rd individual/pair assignment (finding bugs) due next Thursday September 28

4th individual/pair assignment (development technologies) due Tuesday October 3

3rd team assignment (revised proposal) due Thursday October 5

Customer Requirements

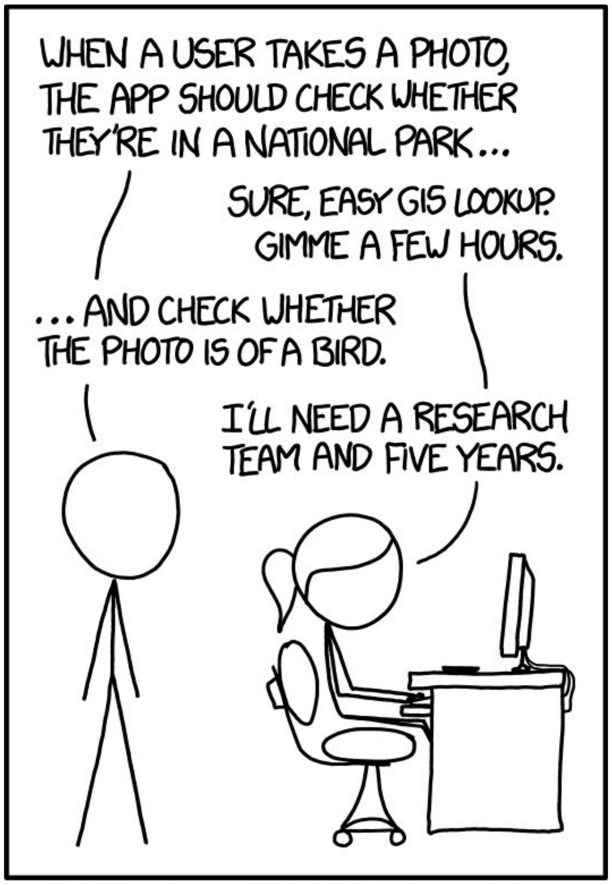
Customers usually have something in mind they want the software to do, of *value* to them

Even in a startup or other new product venture with no customers yet, someone has an *idea* about what will attract customers by providing value (and someone has a business model as to how this will enable paying for it)

W*ho* are the users? - might be used only within customer organization, might also be used by customer's own customers/clients

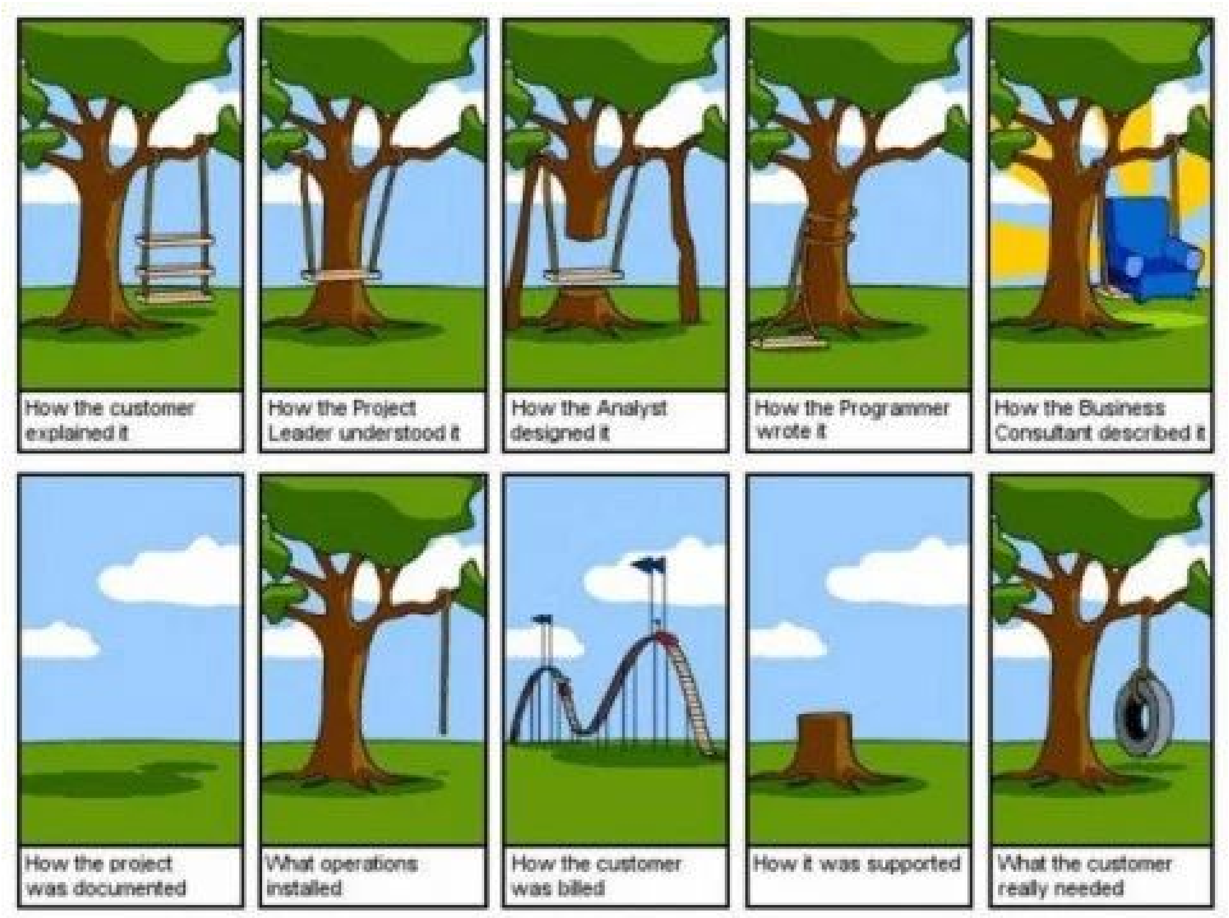
What are the features? What can users do with it that they couldn't do without it, or couldn't do as well without it? Will they want/need to do this?

Is it even feasible?


WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
I'LL NEED A RESEARCH TEAM AND FIVE YEARS.

IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Requirements may be very vague in customer's own mind, or difficult for customer to explain


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Some agile processes include a technically-oriented "product owner" role as a proxy for real customers

Business people may have epic business knowledge, but little technical sophistication and ability to articulate software requirements - and they are unlikely to read any documentation

But the product owner still needs to find out what are the requirements of the actual customer

“Requirements elicitation”

Brainstorming workshops (maybe in form of *design sprints*, covered later on)

Observation and interviews - how will software fit into business workflow, adds constraints on how it interacts with non-software or pre-existing software aspects of that workflow

Role playing - pretend to be the software interacting with the user

Let's do an example requirements elicitation in class: I need a volunteer customer

This customer wants to contract your consulting company to build their software, which is not necessarily a web or mobile app, could be desktop, could be embedded, could be an API...

The class has to elicit the requirements by asking questions of the volunteer customer, who can answer only "Yes", "No", "Sometimes" or "I don't know"

So what are the requirements here?

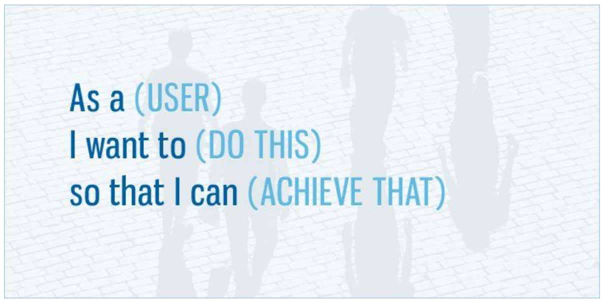
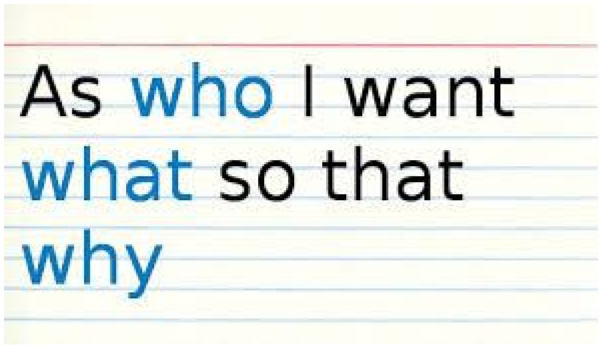
Some requirements cannot easily be expressed as features = non-functional requirements, constraints on design and implementation

- Reliability
- Security
- Performance
- Scalability
- Availability
- Usability
- Accessibility
- Maintainability
- Manageability
- Portability
- Interoperability
- Regulatory

This class we will be primarily concerned with reliability, security and maintainability as non-functional requirements

User Stories

Short simple description of ONE feature, about three sentences that can fit on a 3x5 card

 As a (USER) I want to (DO THIS) so that I can (ACHIEVE THAT)	 As who I want what so that why

“Title or label: As a <type of user>, I want <feature> so that <goal>”

Title or label (omitted from figures) needed for cross-referencing

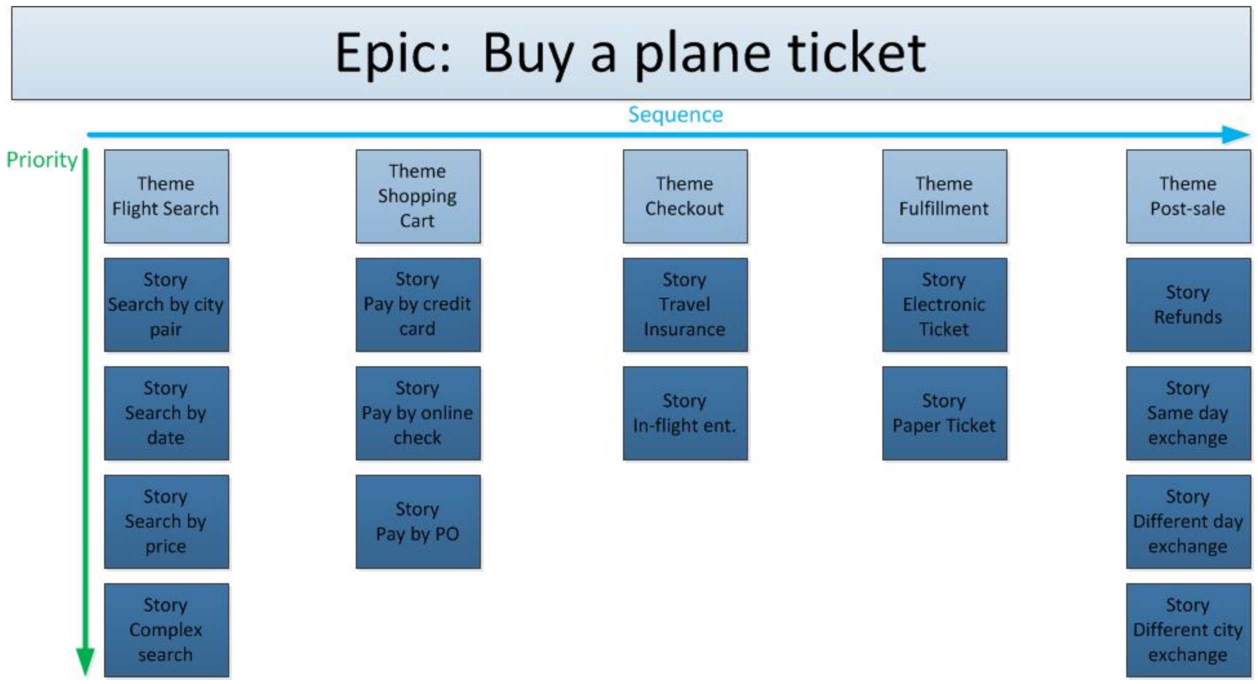
Type of user = role or persona

Customer and developers work together to write the user stories, in customer’s language - usually the developers doing the writing, but customers discuss and approve

“A promise for a conversation” = customer agrees to explain/discuss each user story

Good user stories INVEST - Independent (of other user stories), Negotiable, Valuable, Estimable, Small, Testable

Collections of user stories (or BIG individual user stories) organized into *epics*



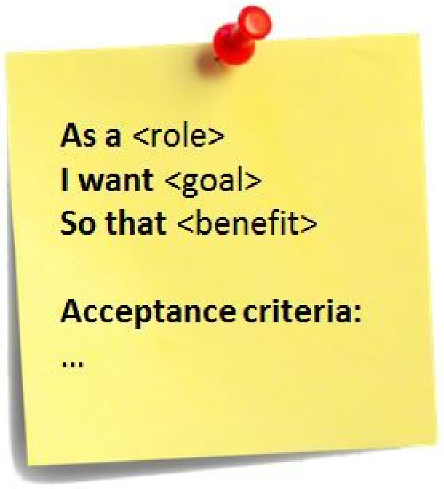
Another example epic: As a user, I want to backup my entire hard drive so that my files are safe

Example breakdown:

> As a typical user, I want to specify folders not to backup so that my backup store isn't filled with things I don't need (alternatively, I want to specify folders to back up)
>
> As a power user, I want to specify files or folders to backup based on file size, date created, date modified so that …
>
> more...

Is this enough to start developing the backup program?

Typically need more details, "acceptance criteria" or "conditions of satisfaction" as high level acceptance tests - all acceptance tests must pass for the customer to accept, and pay for, the software that's why they're called *acceptance* tests



Example user story:

As a marketing executive, I want to select a holiday season to be used when reviewing the performance of past advertising campaigns so that I can identify profitable ones

Example conditions of satisfaction:

List of holidays: Presidents Day, Memorial Day, Labor Day, …

Holidays can span two calendar years: New Years Eve/New Years Day

Holiday seasons can run from one holiday to the next: Thanksgiving through Christmas

Holiday seasons can be set to specific number of days before and specific number of days after the actual holiday

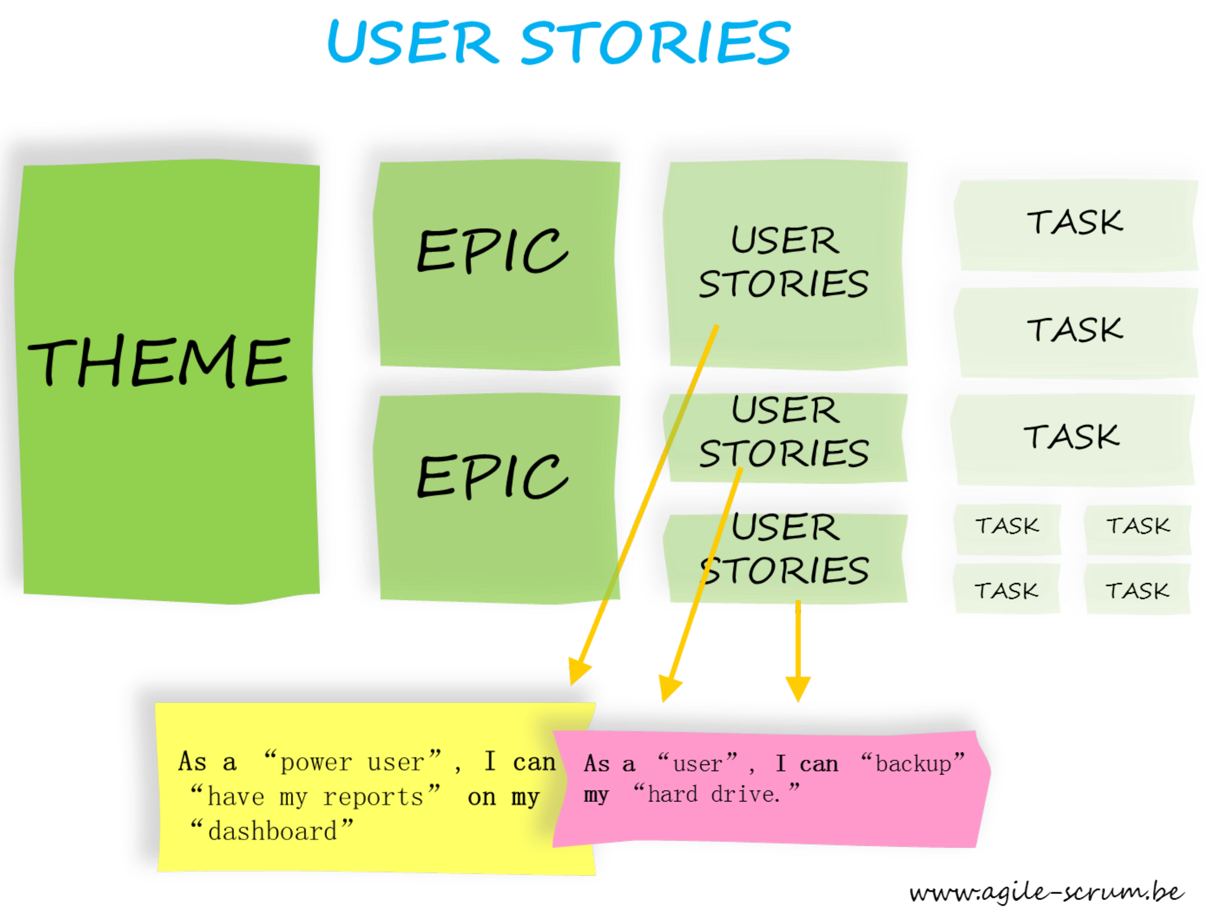
But these won't fit on the 3x5 card!

Right. If you're lucky, maybe they'll fit on the back

But otherwise need to go into an issue tracking or ticket system

Also need tasks (these draw from design, discussed later on)

These have their own 3 x 5 cards, or sticky notes



What is **not** included in user stories?

CRUD - create, read, update, delete (focus on what user wants to do, not underlying database)

Any terminology that the customer does not understand, the customer couldn't have asked for

any technical jargon should be from customer's domain, **not** software engineering

Examples: cache, buffer, third normal form, HTTP

Any specific technology - unless required for interoperability

The customer probably didn't ask for Python or Javascript

But the customer may require that the new application runs on existing hardware, draws data from existing database, interfaces to existing applications, …

When are user stories written?

> Before development starts - story-writing workshop or design sprint
>
> And throughout the project thereafter
> (ideally introduce at the *beginning* of an iteration, not the *middle*)

Do user stories replace requirements document? Yes and No

> Often elaborated into “use cases”, which consider workflow steps, alternative choices, what can go wrong - more realistic level of detail for design
>
> But user stories are great granularity for task boards, burndown charts, prioritization

Priorities come from the customer

> Start with Minimum Viable Product
>
> Then high, medium, low - may change over time
>
> The design may result in dependencies among use cases such that some user stories that seem low priority to customer need to be done before some high priority stories - this may be a bad design but not always (e.g., security should never be added “later”)

Stopped here

Use Cases

User stories and their conditions of satisfaction may not provide enough detail for time estimation (necessary to determine how many user stories will fit in a time-boxed iteration) or to inform design or to break down into development tasks (that can be assigned to developers)

Use cases *expand* user stories to describe the step by step details of how the user role interacts with the software to exercise the feature and achieve the goal or customer value

Also try to capture anything and everything that can *go wrong* (some of these may come from customers, some may be from situations that arise in software that developers know about but customers may not)

Use cases are much longer and more elaborate than user stories

Use case elements (not all present in every use case):

Name - verb/noun or actor/verb/noun communicates scope
(cross-reference to title/label of corresponding user story)

Description - paragraph, might just copy user story

Actors - type(s) of users who engage in activity, could copy role from user story

Preconditions - anything that must be or is expected to be true before use case begins, specify whether can assume true or need to check (with error if not true)

Triggers - if something happens, such as user selects a menu item or the state changes, that causes initiation of the use case

Basic flow (happy path, sunny day path) - set of steps actor needs to take to accomplish goal, with clear description of what system does in response to each step

Alternate flows - less common user/system interactions, special cases

Exception flow - error cases, things that can prevent the user from achieving their goal

Postconditions - anything that *must* be true when the use case ends

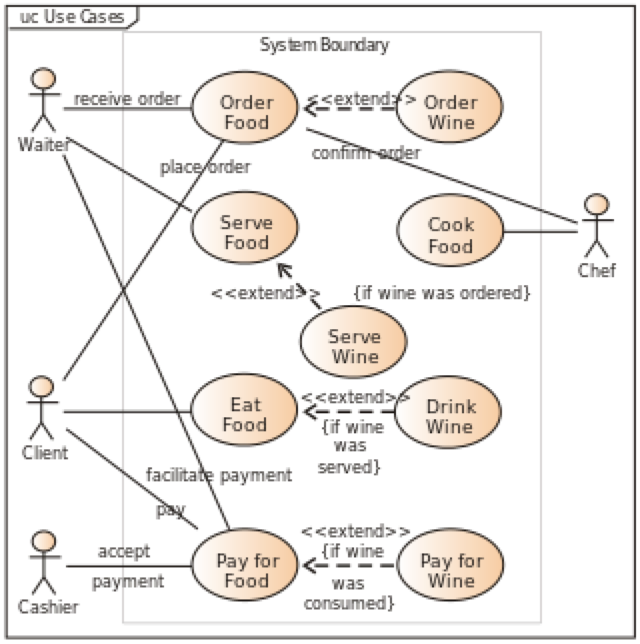
Event - any new triggers (for other use cases) initiated by the completion of the use case

Uses case captures *what* the software needs to do, now *how* the software should do it

What use cases for this course should look like: *heavyweight* version from example use case

We will cover parts of UML (Uniform Modeling Language) later on for "class diagrams", but you do not need to know anything about "use case diagrams" for this course - although future employers may want these

So if you plan to submit anything that looks like this, it's fine but unnecessary



However, this course *will* involve wireframing diagrams - mockups of user interface (soon)