EBU5304 – Software Engineering

Requirements

- Topics
  - Getting started → Why do we need requirements and what are they?
  - Theory and Techniques
    - Functional and non-functional requirements
    - Requirements capture techniques
  - Requirements in Agile process
    - User stories/Story cards
    - Product backlog
    - Prioritisation of stories
    - Estimating

What is a requirement?

- A requirement is a *feature that your system must* have in order to satisfy the customer.
- A requirement is *something your system must do*.

It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

Who provides requirements?

We create software for a reason.
We create software for people.

We need to know what the people want in the software we build.

These are called **REQUIREMENTS**.

1 ______________________

2 ______________________

3 ______________________

4 ______________________

5 ______________________

This is an Intentionally BLANK slide. We will discuss it in class and you can write down notes.

Stakeholder

- Any person or organization who is affected by the system in some way and so who has a legitimate interest
- Stakeholders view the system differently
  - perspective
  - priorities

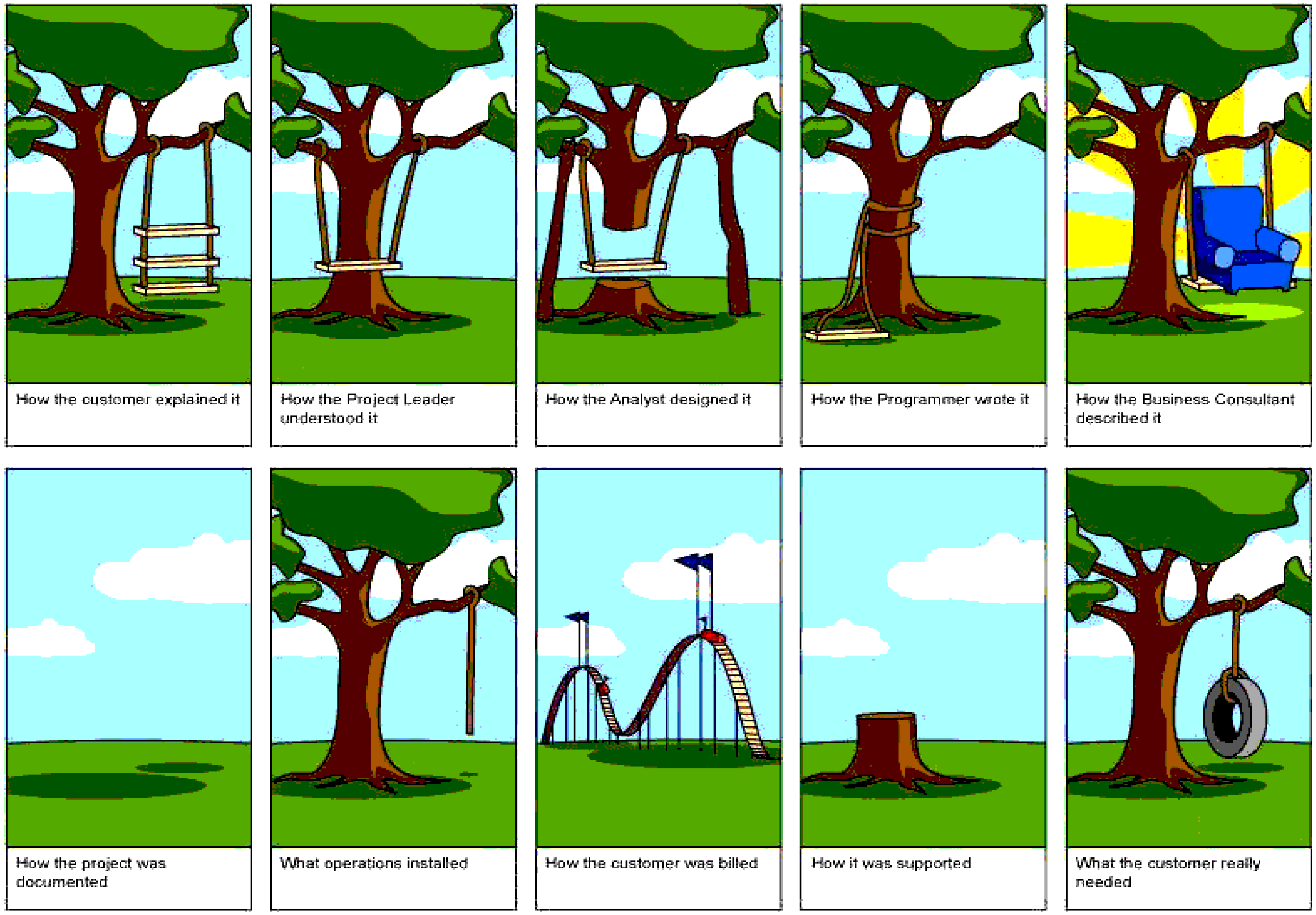
Identify the requirements

- Talk to the customer
- Talk to other stakeholders
- Ask questions
- Brainstorm
- Look at the problem from many points of view

The problems

- Understanding the requirements is among the most difficult tasks that face a software engineer:
  - Does the customer know what is required?
  - Does the end-user have a good understanding of the features and functions that will provide benefit?
  - What they said = what they mean?
  - Good communication?
  - Even if the answers are all yes: requirements will change, and will continue to change throughout the project!

The classic "The Tire Swing" joke


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

Requirements Engineering

- “Solution” of those challenges.
- Helps software engineers to better understand the problem they will work to solve.
- Software engineers and other stakeholders all participate in requirement engineering.
- The <u>extra time invested</u> will save time and money in the long term.

What is a requirement?

- Service and constraint on the system.
- Customers' needs

- Classified as:
  - Functional requirements
    - Services
  - Non-functional requirements
    - Constraints
      - Timing constraints, constraints on the development process, standards, etc.

Functional requirements

- Describe what the system should do – Functionality. For example in a Virtual learning environment (VLE) system:

- The lecturer should be able to set an assignment and deadline.
- The students should be able to submit coursework and view marks.
- The administrator is responsible to enter all the module information and create a student list.
- The student should receive a notification of the announcement.
- The lecturer should be able to provide feedback comments.

Functional requirements problems

- Imprecision
  - Problems arise when requirements are not precisely stated.
  - Ambiguous requirements may be interpreted in different ways by developers and users.
- Should have …
  - Completeness: include descriptions of all facilities required.
  - Consistency: no conflicts or contradictions in the descriptions of the system facilities.
- In practice, it is impossible!

Non-functional requirements

- Define system properties and constraints
  - Reliability, response time, storage requirements.
  - I/O device capability, system representations, etc.
- Process requirements
  - Quality standard, programming language, development method, etc.
- Non-functional requirements may be more critical than functional requirements.
  - *If these are not met*, the system is useless.

Non-functional examples

- Product requirements (product behaviour)

> The user interface should be implemented as simple HTML, without frames or flash.

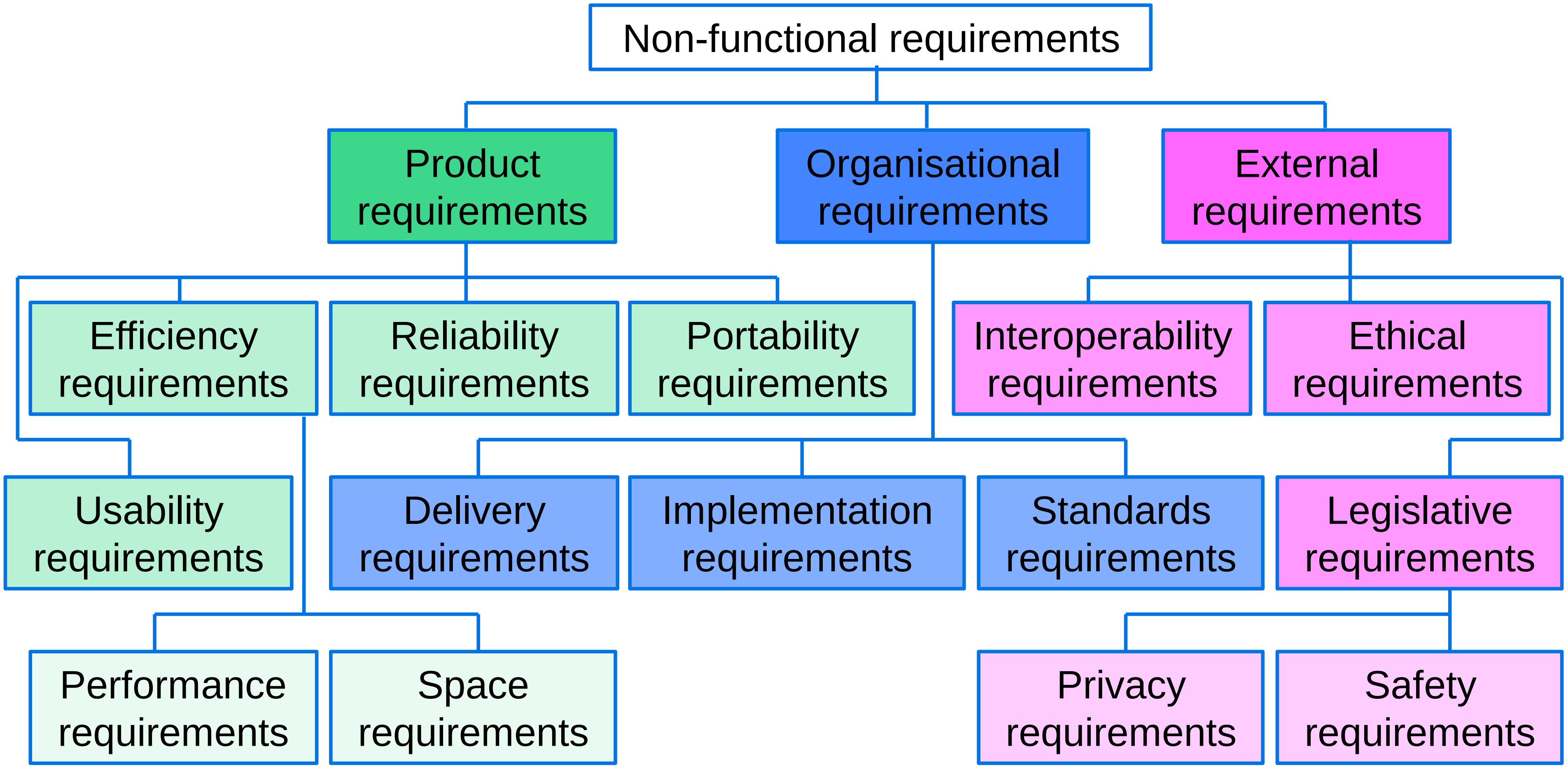
- Organisational requirements (policies and procedures)

> The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.

- External requirements

> The system shall not disclose any personal information about customers, apart from their name and reference number to the operators of the system.

Non-functional requirements



[2] “Software Engineering” by Ian Sommerville, 2001, Addison-Wesley

Non-functional requirements problems

- Difficult to verify
- Verifiable non-functional requirement
  - Using some measure that can be objectively tested.
  - Quantitative
- As developers
  - Convey the intentions of the system users.
  - Translate goals to quantitative.

Example of metrics

- System goal

The system should be easy to use by an experienced administration staff and should be organised in such a way that user errors are minimised.

- Verifiable non-functional requirement

An experienced administration staff shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

Requirements measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	M Bytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Requirements conflicts

- Conflicts between different requirements are common.
- Example: In a digital library system

Requirement A: Item Retrieval

“This system allows the user to retrieve items in any format”.

Requirement B: No File Conversion

“File conversion should not be supported”.

Requirements conflicts

Which is the most critical requirement? There has to be a trade-off between the requirements.

Options as follows:

1. Support file conversions for all major types and increase the budget for the project.
2. Support file conversions for limited set of formats (e.g. .pdf, .rtf and .doc) and increase the budget for the project.
3. Add the requirement: “A separate version of each item, one for every format required, must be submitted to the system when a new item is added”.”.
4. Only support the retrieval of items in formats that are available.

Choose the best option and this has to be agreed by all stakeholders.

The requirements document

- Software Requirements Specification (SRS)
  - The official statement of what is required of the system developers.
  - It is NOT a design document. As far as possible, it should set out WHAT the system should do, rather than HOW it should do it.

Requirements Capture

- Requirements capture:
  - What is to be built?
  - Different starting points for each project, related to what kind of system is required, the type of organisation that requires the system, the technology etc.
- It’s not easy!

Requirements Capture Stages

1. List candidate requirements
   – Modeling the current system or not?
2. Understand the system context, i.e. its scope and purpose
   – What is part of the system and what is outside and interacts with it?
3. Capture functional requirements
4. Capture non-functional requirements

Fact-finding Techniques

- Fact-finding techniques
  - Background reading
  - Interviewing
  - Observation
  - Document sampling
  - Questionnaires

```
This is an Intentionally BLANK slide. We will discuss
it in class and you can write down notes.
```

- "the __________ must be able to read and understand the results of requirements capture. We must use the ____________________________ to describe these results".

[3] "The Unified Software Development Process", by Jacobson *et al*, 1999 Addison-Wesley

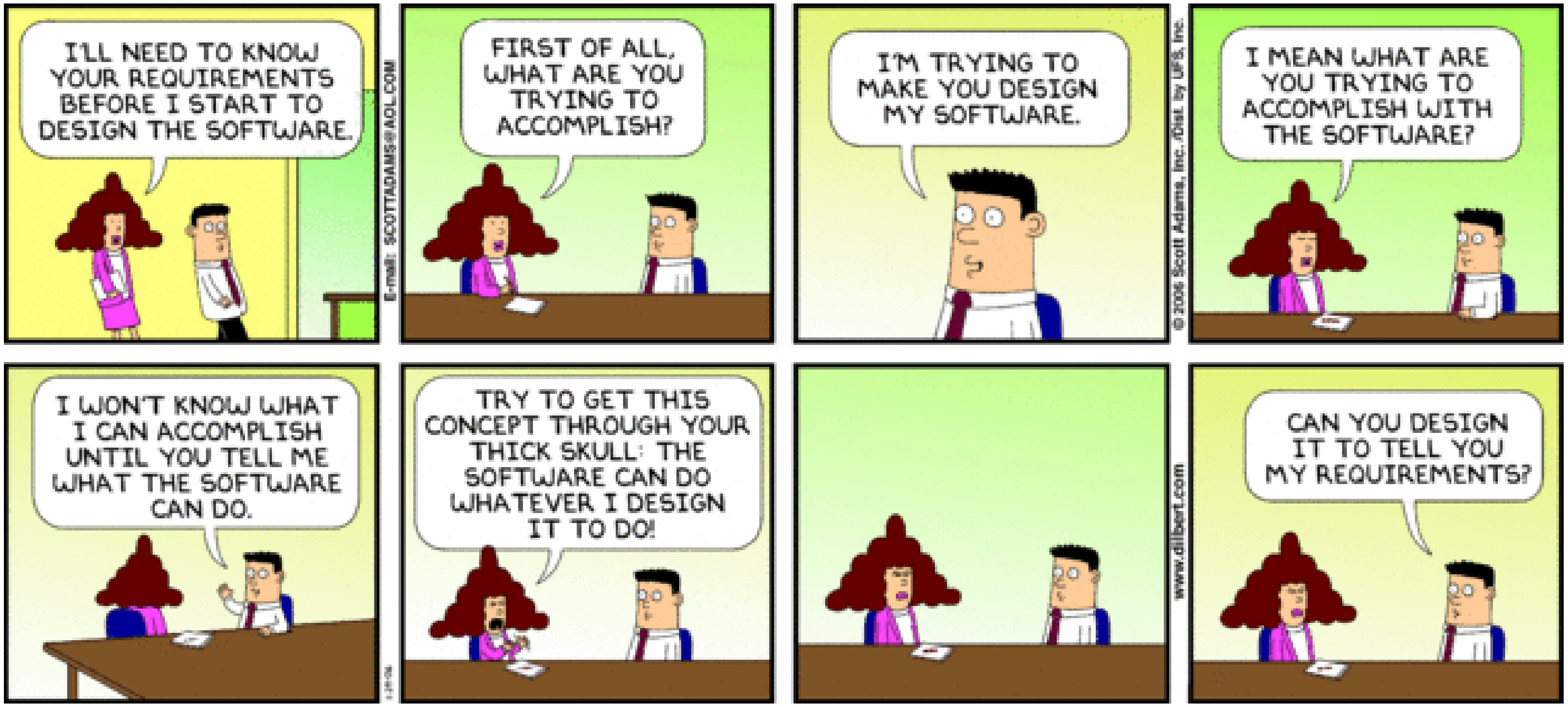
1. Background Reading

- Learn more about the company and department the system is for, using:
  - Company reports
  - Organisation charts
  - Policy manuals
  - Job descriptions
  - Documentation of existing system
- Why do we do this?

2. Interviewing

- Most widely used technique
  - Who?
  - Structured meeting, may be 1-1 or a group.
  - Identify the roles that you want to interview, not the people.
  - Concentrate on identifying the tasks that users complete.
- Start with open-ended questions like
  - "Could you tell me what you do?"
  - "Why", if the developer is an expert
- Move on to more specific questions, to obtain specific information about tasks.
- Direct Contact, Open Mind
- Collect samples of documents that are used in the current system.

Not like this…


I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?
E-mail: SCOTTADAMS@AOL.COM
© 2006 Scott Adams, Inc./Dist. by UFS, Inc.
www.dilbert.com

http://www.2lucu.com/funny-software/attachment/1125/

3. Observation

- Developer observes the user using the current system (manual or automated).
- Provides the developer with a better understanding of the system.
- Often during an interview, the user will concentrate on the everyday tasks and will forget about the "one-off's".
  - These can be incorporated into the new system if the developer observes them first hand.
- Observation refers to "planned watching of an area under study"
  - Findings can be distorted if the user behaves differently.
- Useful after interviews if there is some confusion over tasks
  - Open-ended where developer just wishes to observe tasks etc.
  - Closed, and related to those tasks that caused confusion in the interviews.
- However, ideal for monitoring tasks from start to finish through a department or company.

4. Document or Record Sampling

- A specialist form of observation.
- Different focuses:
  - collect copies of documentation
  - obtain details
  - identify patterns in requirements for the system
- Developer should be aware that the existing forms may not be the best way to model the new system.
- Ideal for capturing non-functional requirements.

5. Questionnaires

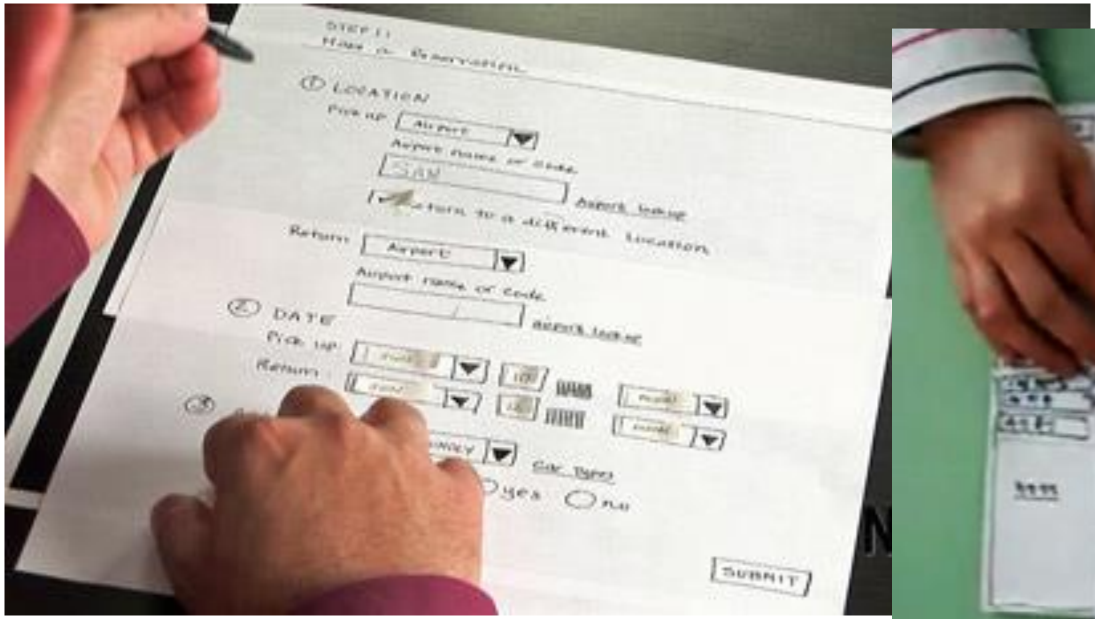
- “The aim of questionnaire design is to pose questions where misinterpretation is not possible and there is no bias”.
- Economical way of gathering data, ideal for the development of systems to be used by multiple users on multiple sites.
- Results could be analysed by a computer.
- Often suffer from “questionnaire fatigue” and response is therefore low or inaccurate.
- Difficult to write a good questionnaire.

Prototype user interface

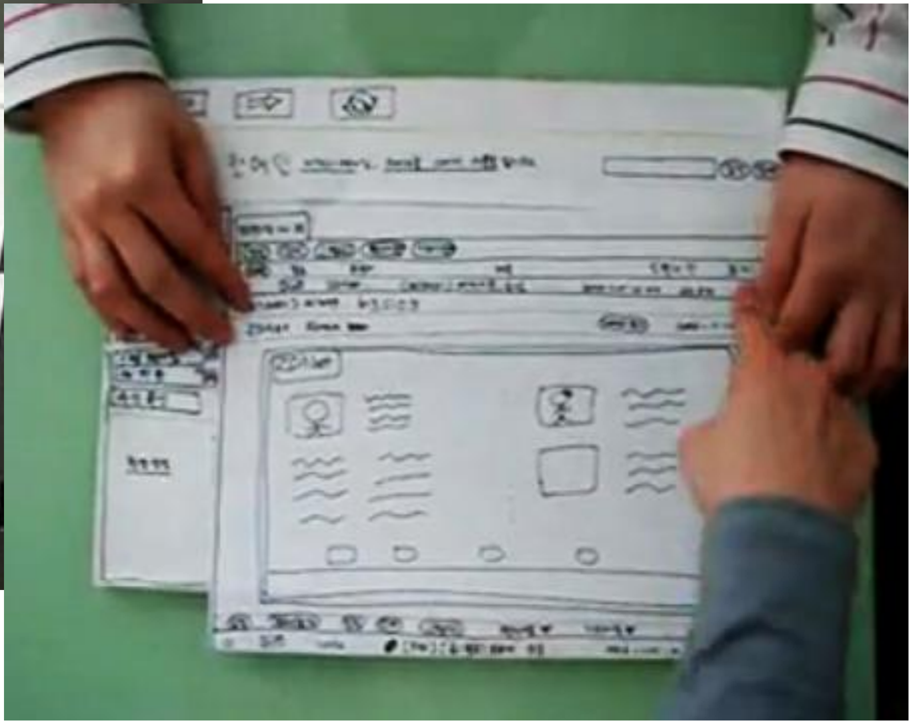
- Physical user-interface design
  - Draw the user interfaces on a paper
    - GUI?
    - Get quick feedback
- Logical user-interface design (information)
  - Which user-interface elements are needed?
  - How are these elements related to each other?
  - What should they look like?
  - How should they be manipulated?

Paper prototyping

- Paper prototyping is a technique that allows you to create and test user interfaces quickly and cheaply. It's easier to change a prototype than the final design.



http://www.nngroup.com/



http://cosent.nl/en/blog/agile-design-paper-prototyping

Requirements in Agile process

User Stories

- In Agile process, user requirements are expressed as **user stories**.
  - helps shift the focus from writing about requirements to talking about them.
- These are written on cards, called **story cards**.
- All agile user stories include a written sentence or two and, more importantly, a series of conversations about the desired functionality.

User Stories

- The **customer** chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.
- The **development team** break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.

Story Cards

- **User stories** are short, simple description of a feature They typically follow a simple template:

As a <*type of user*>,
I want <*some goal*>
so that <*some reason*>.

Story Cards

- **User stories** examples:

As a user,
I want to backup my entire hard drive
so that I won't lose any work.

As a Website administrator,
I want to be able to run checking reports every 15 minutes
so that I would have the chance to see when the application is approaching a threshold that will slow down the system.

?

Story Cards

- **User stories** examples:

As a concerned mother,

I want to receive weather alerts for areas where my children live

so that I know when I need to worry and call them to make sure they are safe.

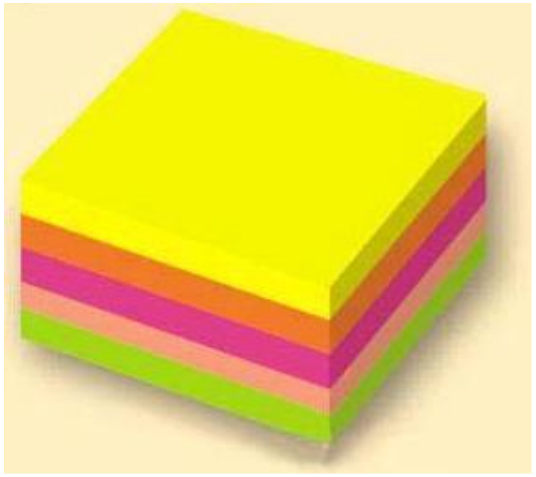
As a busy executive,

I want to be able to save favourites on my mobile weather application

so that I can choose from a finite drop-sown list to easily locate the weather in the destination I am travelling to.

Story cards and story wall

- User stories are often written on index cards or sticky notes, stored in a shoe box, and arranged on walls or tables to facilitate planning and discussion.
- As such, they strongly shift the focus from writing about features to discussing them. In fact, these discussions are more important than whatever text is written.



Story cards and story wall

Daily team stand-up meeting

Story sorting

Epics

- User stories is that they can be written at varying levels of detail.
- Large user stories are generally known as epics.
- Because an epic is generally too large for an agile team to complete in one iteration, it is split into multiple smaller user stories before it is worked on.

Epics break down example 1

Epic

As a user,

I want to backup my entire hard drive

so that I won’t lose any work.

Break down story 1

As a power user,

I want to specify files or folders to backup based on file size, date created and date modified

So that I can manage the files better.

Break down story 2

As a user,

I want to indicate folders not to backup

So that my backup drive isn't filled up with things I don't need.

Epics break down example 2

Epic

As a busy executive,

I want to be able to save favourites on my mobile weather application

so that I can choose from a finite drop-sown list to easily locate the weather in the destination I am travelling to.

Break down story 1

As a busy executive,

I want to be able to save a search location to my list of favourites from my mobile device

So that I can reuse that search on future visits.

Break down story 2

As a busy executive,

I want to be able to name my saved searches from my mobile device

so that I know how to access each saved search in the future.

Epics break down example 2

Break down story 3

As a busy executive,

I want the name of my saved searches to default to the city name, unless I choose to manually override it

So that I can streamline saving my searches.

Break down story 4

As a busy executive,

I want my “saved favourites” option on the user interface to be presented on a mobile device near the “search location”

So that I have the option of starting a new search or using a saved one, and can minimise my keystrokes within the weather application.

Acceptance Criteria

- Acceptance Criteria is simply a high-level acceptance test that will be true after the agile user story is complete.
- Normally written on the back of the story card.
- It is a great way to ensure that a story is understood and to invite negotiation with the team about the business value that we are trying to create.

Acceptance Criteria examples

Story

As a vice president of marketing,

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.

Acceptance Criteria

- Make sure it works with major retail holidays: Christmas, Easter, Mother's Day, Father's Day, New Year's Day.
- Holiday seasons can be set from one holiday to the next (such as Christmas to New year's Day).
- Holiday seasons can be set to be a number of days prior to the holiday.

Acceptance Criteria examples

Story

As a busy executive,

I want the name of my saved searches to default to the city name, unless I choose to manually override it

So that I can streamline saving my searches.

Acceptance Criteria

- Verify that the correct city name is auto populated when the search found the location by *city* and the Save option is chosen from a mobile device.
- Verify that the correct city name is auto populated when the search found the location by *postcode* and the Save option is chosen from a mobile device.
- Verify that the default location name is saved to the database.
- Verify that the default location name is populated as a saved location in future visits.
- Verify that the location name can be manfully over written if the default location name is not desired.

Non-functional Requirements as User Stories

- Non-functional requirements (constraints) can also be handled as user stories. Examples:
  - As a customer, I want to be able to run your product on all versions of Windows from Windows XP on, so that we can make use of older PCs.
  - As the CTO, I want the system to use our existing orders database rather than create a new one, so that we don't have one more database to maintain.
  - As a user, I want the site to be available 99% of the time I try to access it, so that I don't get frustrated and find another site to use.
  - As someone who speaks Chinese, I want to have a Chinese version of the software.
- If you can't find a way to word the constraint, just write the constraint in whatever way feels natural.

Story card template (front)

Story name	**Story ID**
As a **I want** **So that**	
Priority high 1 2 3 4 5 low **Date Started**	**Iteration number** **Date Finished**

The template is available to download from QMPLUS

Story card template (back)

Acceptance Criteria
Notes

The template is available to download from QMPLUS

Who write user stories?

- Anyone can write user stories.
- Over the course of a good agile project, you should expect to have user story examples written by each team member.
- Who writes a user story is far less important than who is involved in the discussions of it.

Not like this…

I'LL GO UP AND FIND OUT WHAT THEY NEED AND THE REST OF YOU START CODING!
SKELTON

`http://www.cs.uni.edu/~mccormic/humor.html`

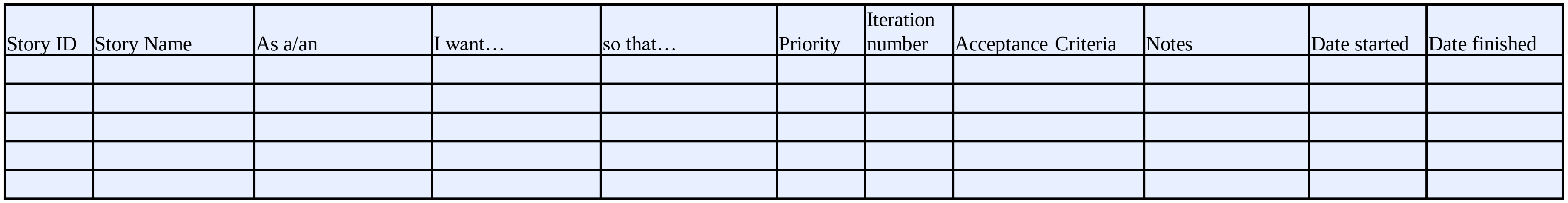
When are user stories written?

- User stories are written throughout the agile project.
- Usually a story-writing workshop is held near the start of the agile project. Everyone on the team participates with the goal of creating a product backlog that fully describes the functionality to be added over the course of the project (or a three to six-month release cycle) within it.
- Epics will later be decomposed into smaller stories that fit more readily into a single iteration. Additionally, new stories can be written and added to the product backlog at any time and by anyone.

Product backlog

- Product backlog, which is a prioritised list of the functionality to be developed in a product or service.
- User stories have emerged as the best and most popular form of product backlog items.
- Important: the written part of an agile user story ("As a …, I want …") is incomplete until the discussions about that story occur.
- It's often best to think of the written part as a pointer to the real requirement.

Product backlog (excel template)



Story ID	Story Name	As a/an	I want…	so that…	Priority	Iteration number	Acceptance Criteria	Notes	Date started	Date finished

The template is available to download from QMPLUS

Prioritisation of stories

- Product backlog user stories must be prioritised.
  - Based on business value
  - Value must also be supported by a positive return on investment (ROI)
  - Consideration of the risks

MoSCoW

- A method of prioritisation favoured by the DSDM (dynamic systems development method). Elements are:
  - Must have: features must be implemented, and if not, the system would not work.
  - Should have: features are important but can be omitted if time or resources constraints appear.
  - Could have: features enhance the system with greater functionality, but the timelines of their delivery is not critical.
  - Want to have: features serve only a limited group of users and do not drive the same amount of business value as the preceding items.

MoSCoW example

- Payment processing story:
  - Must have: ability to accept Visa and MasterCard.
  - Should have: add American Express.
  - Could have: add PayPal.
  - Want to have: add gift cards.

Estimating

- Estimates: how long the work is likely to take
- Some methods include:
  - Level of Effort or T-shirt sizing: small, medium, large, extra large…
  - Ideal Time: under ideal circumstance
  - Hours: actual clock hours
  - Story points: an arbitrary measure that allows the teams to understand the size of the effort

Estimating story points

- Story points:
  - Fibonacci Sequence: 1,2,3,5,8,13,21…
  - As the points get higher, the degree of uncertainty is increasing.
  - If a story is estimated at greater than 13 story points, it might still be an epic and needs to be broken down.
  - Abstract and negotiable
  - Everyone on the team agrees to the size of a story point, using relative sizing
    - E.g. adding a field to the database, agreed as a 5 story points

Good user story?

- INVEST
  - **I**ndependent
  - **N**egotiable
  - **V**aluable
  - **E**stimatable
  - **S**mall
  - **T**estable

Summary

- Requirements
  - Why do we need requirements and what are requirements?
  - Theory and Techniques
    - – Functional and non-functional requirements
    - – Requirements capture techniques
  - Requirements in Agile process
    - User stories/Story cards
    - Product backlog
    - Prioritisation of stories
    - Estimating

References

- Chapter 4 – "Software Engineering" textbook by Ian Sommerville
- Chapters 2+3 – "Head First Object Oriented Analysis &Design" textbook by Brett McLaughlin et al
- User stories by Mike Cohn http://www.mountaingoatsoftware.com/agile/
- Introduction to Agile by Sondra Ashmore