REQUIREMENTS ENGINEERING

DR. VIVEK NALLUR

VIVEK.NALLUR@SCSS.TCD.IE

https://www.scss.tcd.ie/vivek.nallur/teaching/slides/

OUTLINE OF THIS TALK

- What are requirements? (and why do we need to engineer them?)
- What does requirements engineering involve?
- First two steps in requirements engineering
- Your first assignment!

WHAT IS A 'REQUIREMENT' ANYWAY?

According to the *IEEE*:

- A condition or capability needed by a user to solve a problem or achieve an objective.
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formal document

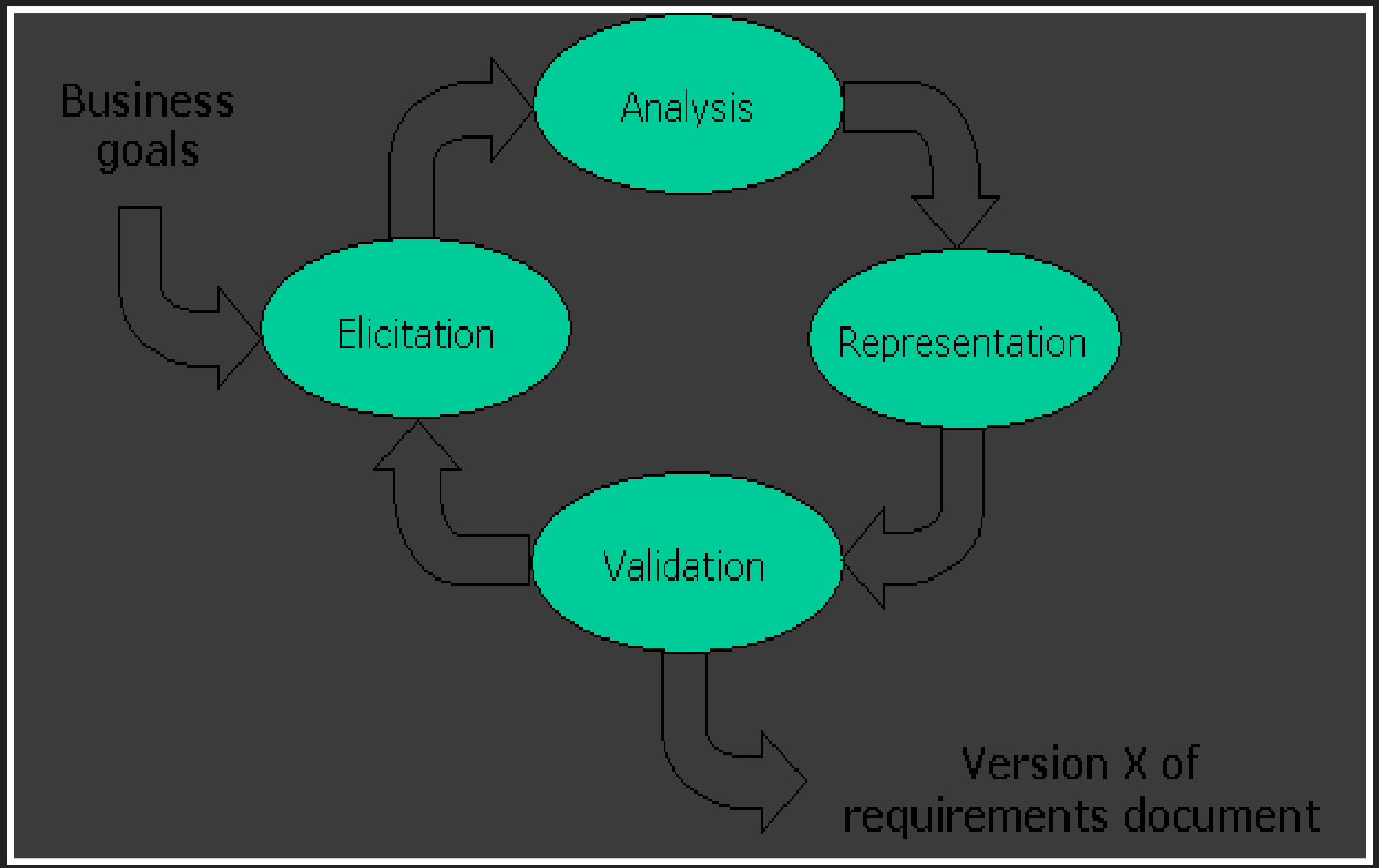
Requirements cover not only the desired functionality of a system or software product, but also address:

- *Non-functional issues* (*e.g.*, performance, reliability, etc.)
- *Constraints on design* (*e.g.*, must operate with existing hardware/software)
- *Constraints on implementation* (*e.g.*, must be written in Java)

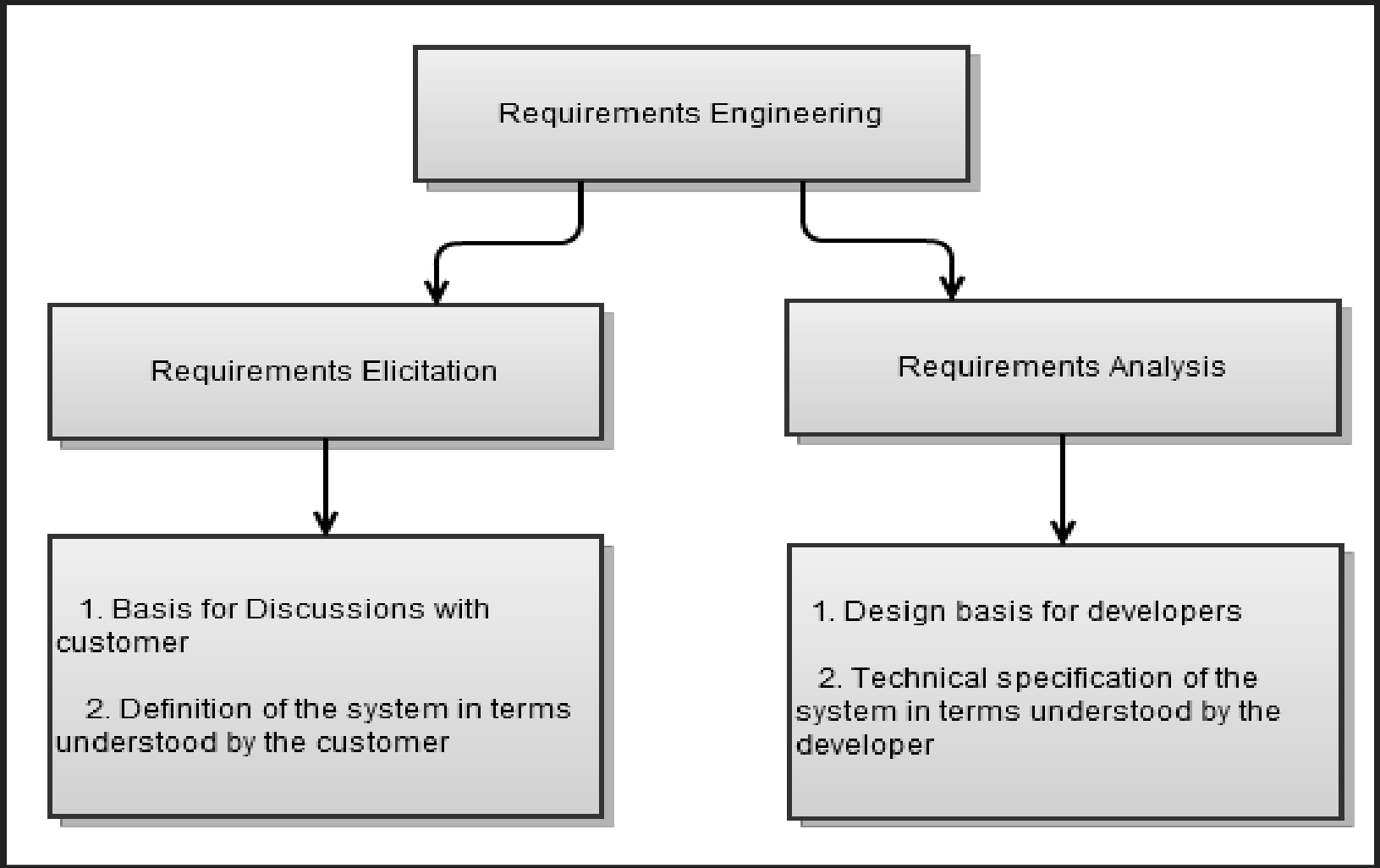
THE ELEMENTS OF REQUIREMENTS ENGINEERING

- Elicitation
- Analysis
- Representation
- Validation

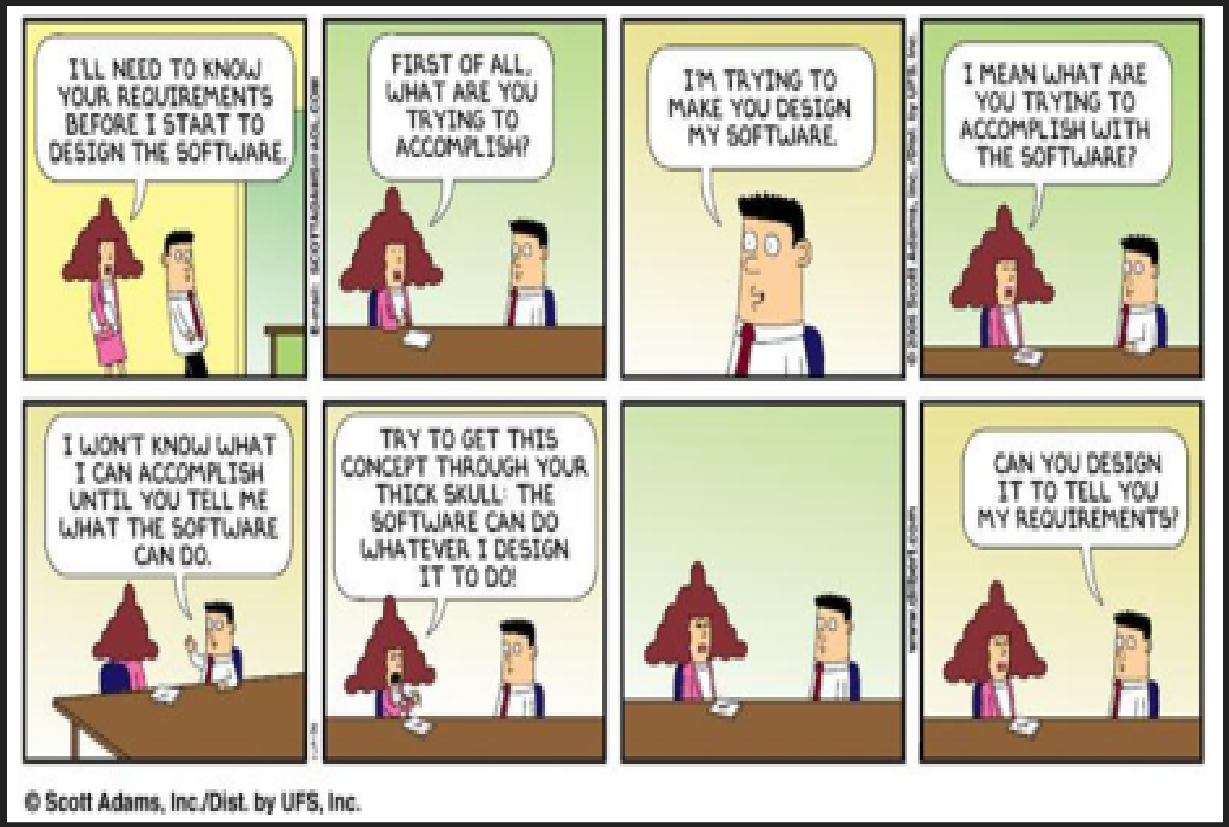
NOT A LINEAR PROCESS



TWO DIFFERENT SET OF STAKEHOLDERS



ELICITATION: TALKING TO THE CUSTOMER

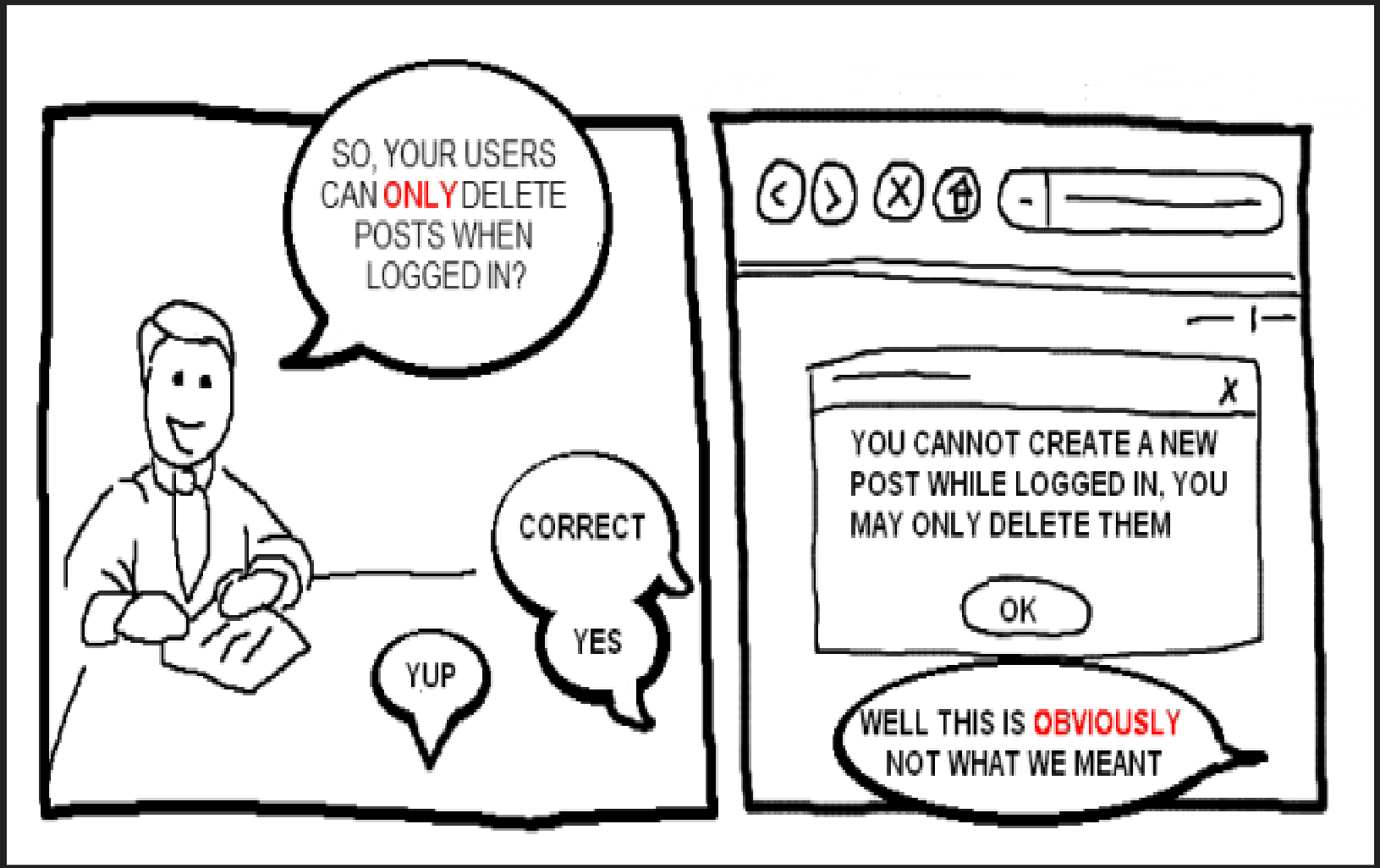


ELICITATION TECHNIQUES

Technique	Good for	Kind of data	Plus	Minus
Questionnaires	Answering specific questions	Quantitative and qualitative data	Can reach many people with low resource	The design is crucial. Response rate may be low. Responses may not be what you want
Interviews	Exploring issues	Some quantitative but mostly qualitative data	Interviewer can guide interviewee. Encourages contact between developers and users	Time consuming. Artificial environment may intimidate interviewee
Focus groups and workshops	Collecting multiple viewpoints	Some quantitative but mostly qualitative data	Highlights areas of consensus and conflict. Encourages contact between developers and users	Possibility of dominant characters
Naturalistic observation	Understanding context of user activity	Qualitative	Observing actual work gives insight that other techniques cannot give	Very time consuming. Huge amounts of data
Studying documentation	Learning about procedures, regulations, and standards	Quantitative	No time commitment from users required	Day-to-day work will differ from documented procedures

Source: Preece, Rogers, and Sharp "Interaction Design: Beyond human-computer interaction", p214

THE PROBLEM WITH TALKING (IN NATURAL LANGUAGE)



THE PROBLEM WITH MAKING THE USER WRITE IT DOWN

Five Horrible Requirements

#1 – "The system must have good usability"

#2 – "Response time should be less than 2 seconds"

#3 – "Round-the-clock availability"

#4 – The system shall work just like the previous system but on platform X

#5 – The system has to be bug-free

BAD REQUIREMENTS: AMBIGUOUS

REQ1 The system shall not accept passwords longer than 15 characters.

It is not clear what the system is supposed to do:

- The system shall not let the user enter more than 15 characters.
- The system shall truncate the entered string to 15 characters.
- The system shall display an error message if the user enters more than 15 characters.

The corrected requirement reflects the clarification:

REQ1 The system shall not accept passwords longer than 15 characters. If the user enters more than 15 characters while choosing the password, an error message shall ask the user to correct it.

BAD REQUIREMENTS: NON-ATOMIC

REQ1 The system shall provide the opportunity to book the flight, purchase a ticket, reserve a hotel room, reserve a car, and provide information about attractions.

This requirement combines five atomic requirements, which makes traceability very difficult. Sentences including the words “and” or “but” should be reviewed to see if they can be broken into atomic requirements.

BAD REQUIREMENTS: LONG AND WINDING SENTENCES

REQ1 Sometimes the user will enter Airport Code, which the system will understand, but sometimes the closest city may replace it, so the user does not need to know what the airport code is, and it will still be understood by the system.

This sentence may be replaced by a simpler one:

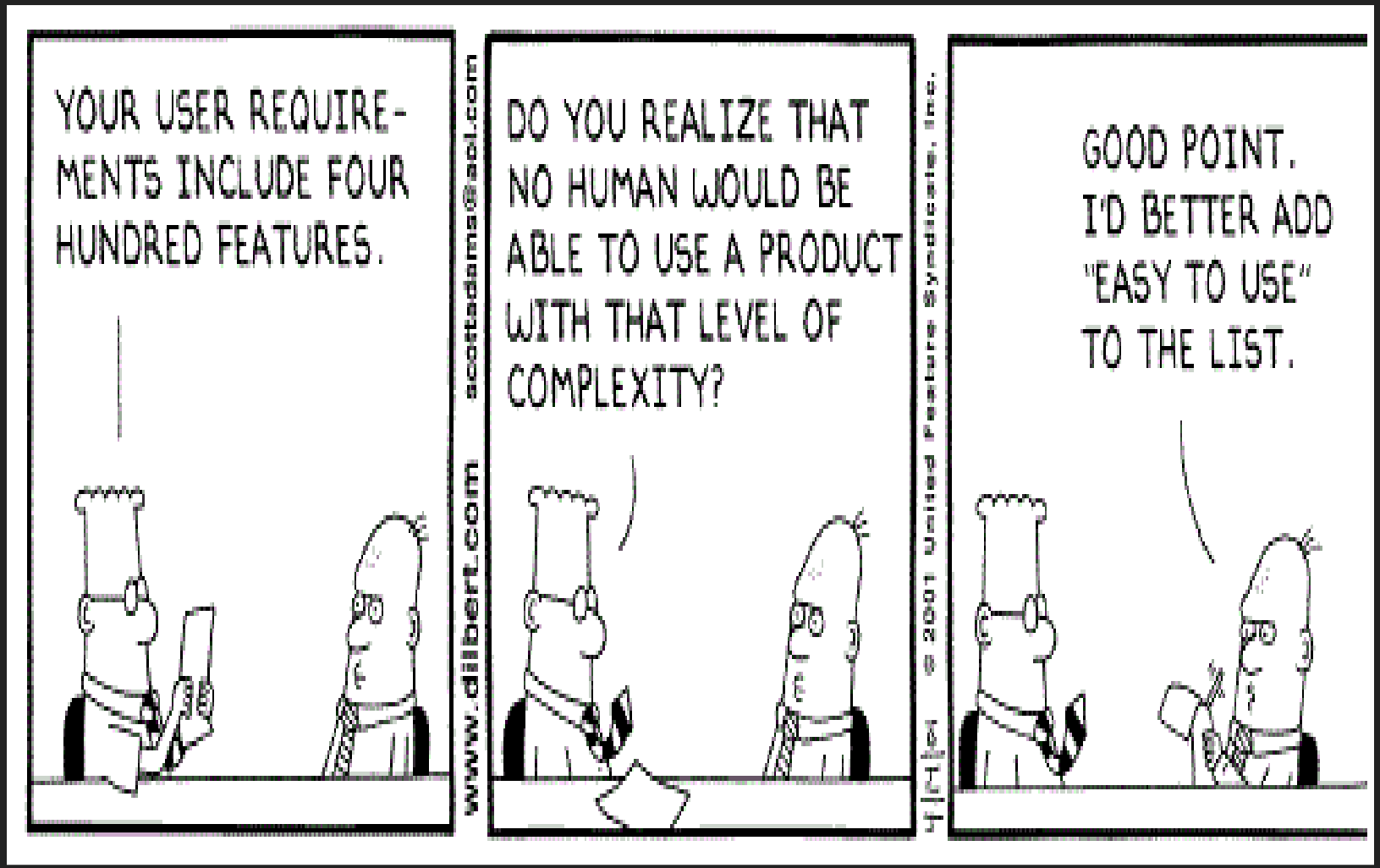
REQ1 The system shall identify the airport based on either an Airport Code or a City Name.

BAD REQUIREMENTS: NON-INDEPENDENT

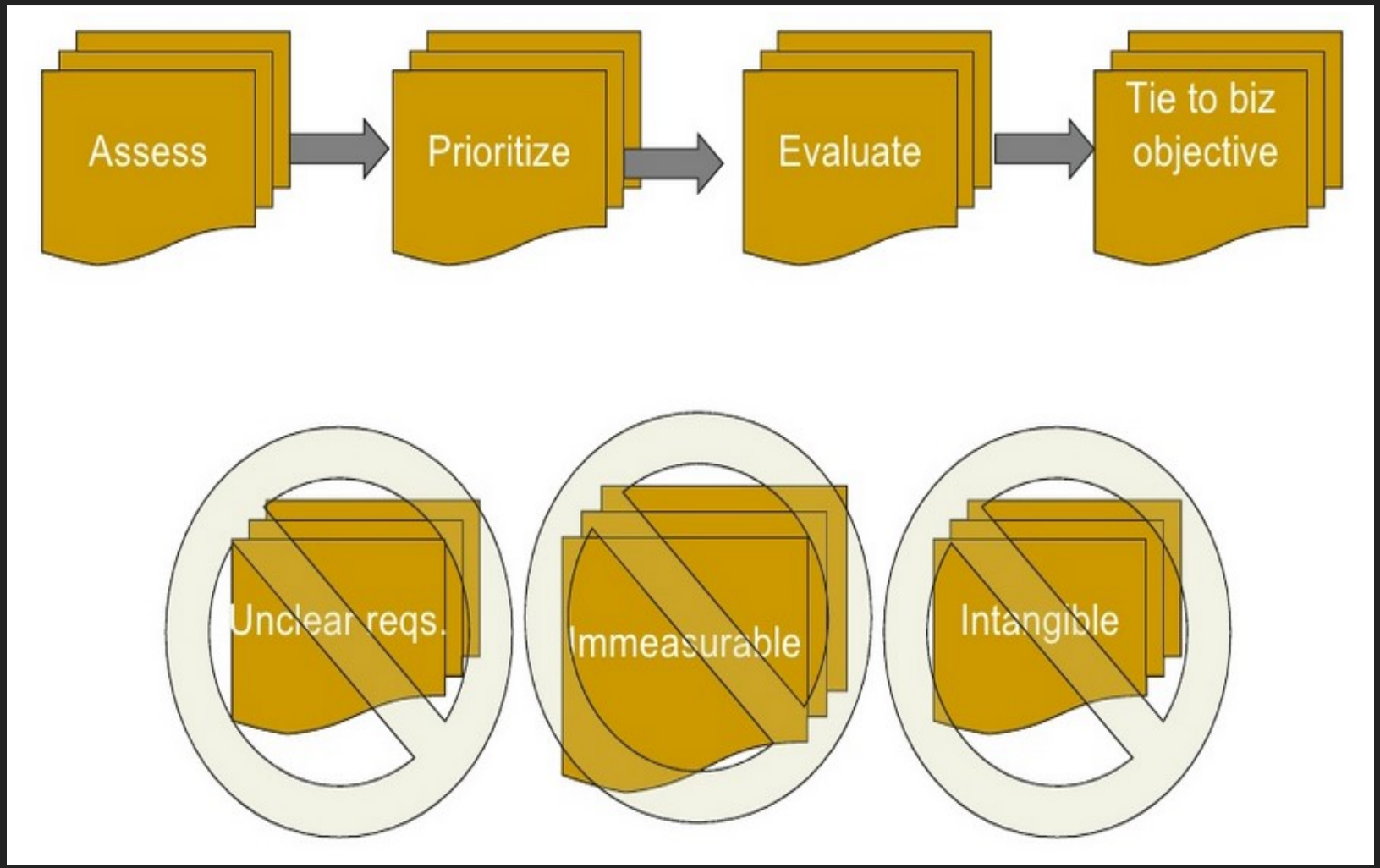
REQ1 The list of available flights shall include flight numbers, departure time, and arrival time for every leg of a flight.

REQ2 It should be sorted by price.

GOT ALL THE REQUIREMENTS?



REQUIREMENTS ANALYSIS



REQUIREMENTS OF REQUIREMENTS

- Clear
- Measurable
- Concise
- Feasible
- Necessary
- Prioritized

ASSIGNMENT FOR THE WEEK

- Download problem statement from: https://www.scss.tcd.ie/Vivek.Nallur/teaching/cs3012
- Download Engineer's Log from the same location
- Submission deadline: **10-October-2016, 10:00 a.m.**

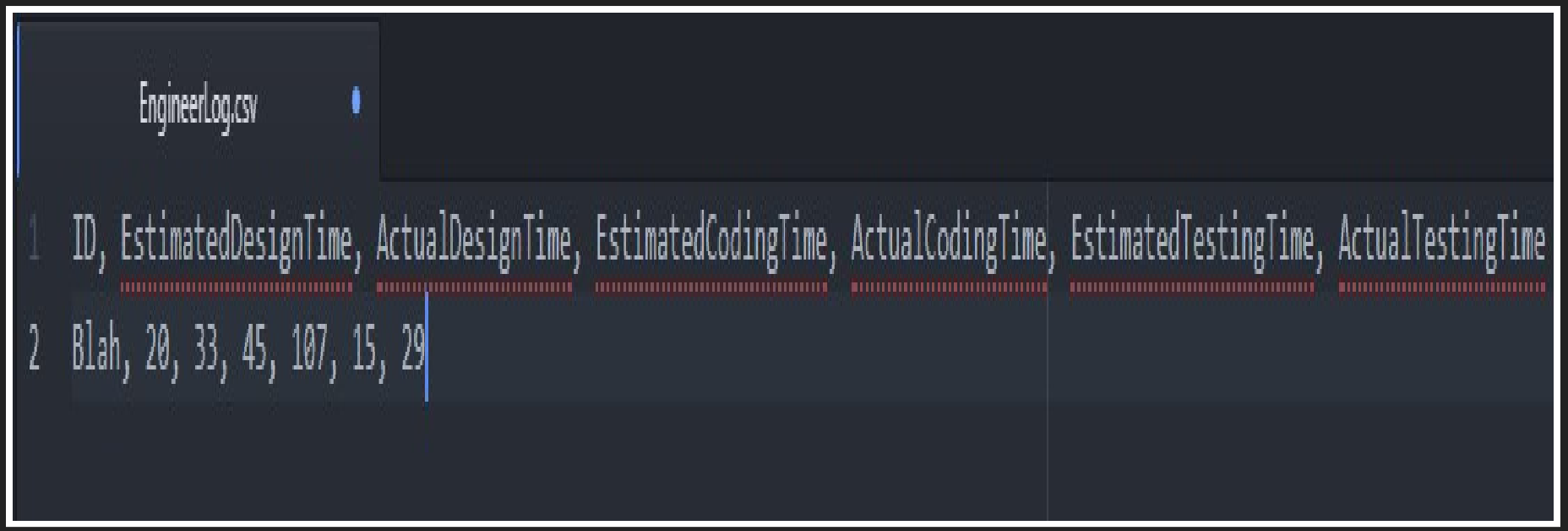
YOUR PROCESS - 1

- Keep an Engineer's Log *throughout* your assignment work
- For each assignment, there are three tasks:
  - Design your solution
  - Code it
  - Test and fix bugs

YOUR PROCESS - 2

- For each task:
  - Estimate how much time (in minutes) it will take, *before* the task.
  - Record estimate in the Engineer's Log
  - Do the task
  - Record the actual time taken

SAMPLE LOG


EngineerLog.csv
1 ID, EstimatedDesignTime, ActualDesignTime, EstimatedCodingTime, ActualCodingTime, EstimatedTestingTime, ActualTestingTime
2 Blah, 20, 33, 45, 107, 15, 29

YOUR PROCESS - 3

- Each one of you has a personalized repository: https://gitlab.scss.tcd.ie/vivek.nallur/cs3012_[your-user-id]/
- For each assignment:
  - Create a directory for it (e.g., 1, 2, 3,...)
  - Check-in your code (**only** the .java file(s))
  - Check-in your Engineer's Log

GRADING OF ASSIGNMENTS - 1

- The ***Input*** and the ***Output*** specification will be checked strictly
- Every mis-match will be counted as a defect
- Your programming score will be:
  $Number\ of\ Test\ Cases - Number\ of\ Defects$
- *Fair warning*: A plagiarism detector will be used!

GRADING OF ASSIGNMENTS - 2

From your Engineer's Log:

$$Velocity_{task} = \frac{ActualTimeforTask}{EstimatedTimeforTask}$$

$$AverageVelocity = \frac{\sum Velocity_{task}}{NumberofTasks}$$

GRADING OF ASSIGNMENTS - 3

- Each individual assignment is worth 10 marks
- Correctness --> 5 marks
  ($Number\ of\ Test\ Cases - Number\ of\ Defects$)
- Predictability --> 5 marks
  ($AverageVelocity$ normalized to a *z-score*. Top 20% gets 5 marks, next 20% get 4, and so on)

THAT'S ALL FOLKS!

Any questions?