CSC 7003 : Basics of Software Engineering

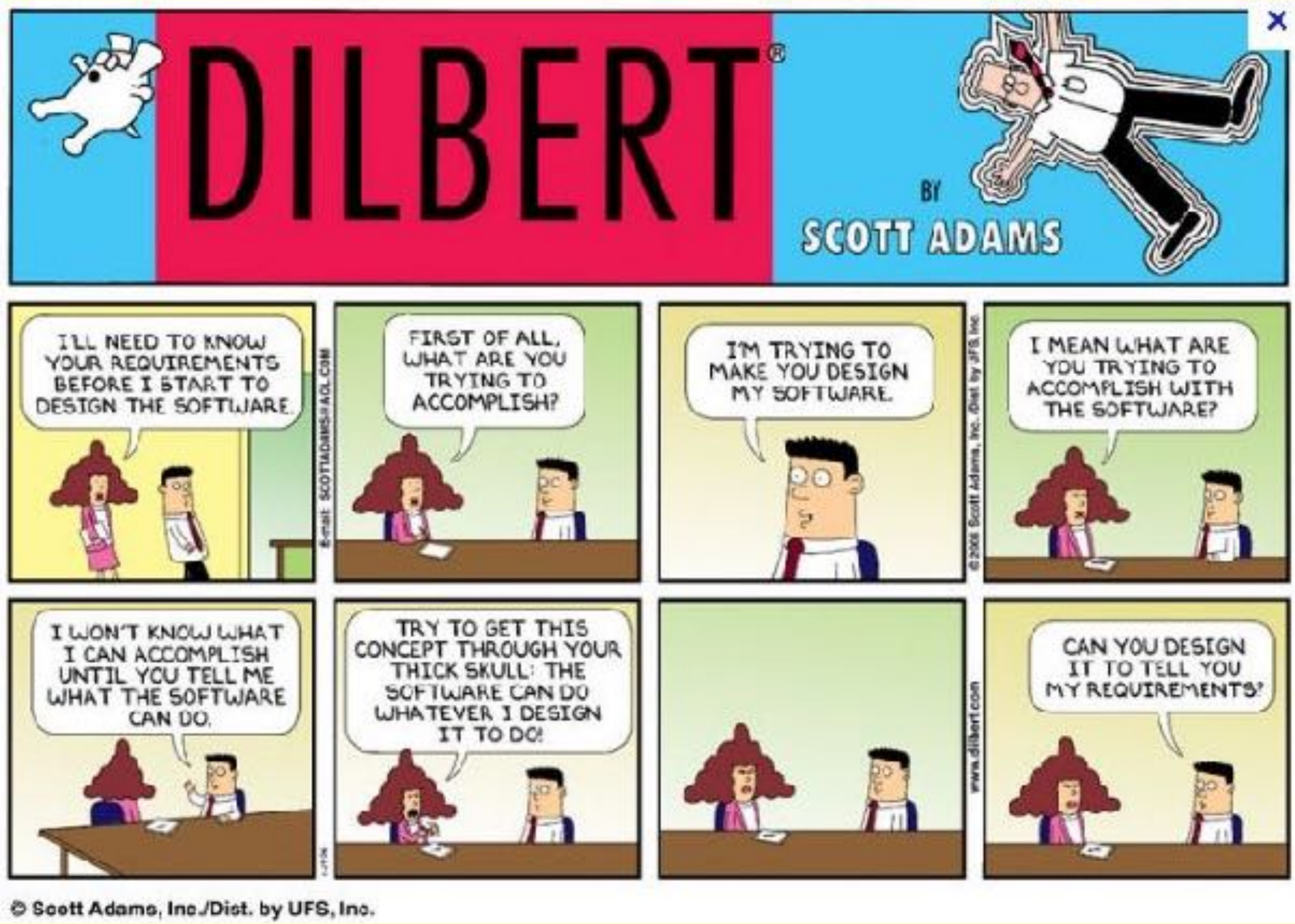
J Paul Gibson, D311

paul.gibson@telecom-sudparis.eu

http://www-public.telecom-sudparis.eu/~gibson/Teaching/CSC7003/

Requirements Engineering

/~gibson/Teaching/CSC7003/L3-Requirements.pdf


DILBERT®
BY
SCOTT ADAMS
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?
www.dilbert.com
© Scott Adams, Inc./Dist. by UFS, Inc.

Requirements modelling is important in all life cycles

Requirements should

- say *what* not *how*
- be *customer oriented*
- be *consistent*
- be *complete*
- be *unambiguous*
- be *useful to designers*

Requirements capture and validation is probably the most difficult part of software engineering. It is also one of the most critical parts

Reading Material

Requirements engineering in the year 00: A research perspective, A van Lamsweerde, 2000

Requirements Engineering: A Roadmap, Bashar Nuseibeh and Steve Easterbrook, 2000

On Non-Functional Requirements in Software Engineering, Lawrence Chung and Julio Cesar Sampaio do Prado Leite, 2009

Requirements Engineering, Elizabeth Hull, Ken Jackson and Jeremy Dick, 2005

Requirements: the issues

The world of software engineering cannot always agree on requirements modelling:

- formal or informal
- operational or logical
- textual or graphic
- client-led or analyst-led

Requirements: the issues

My guidelines:

- make the model as ‘formal’ as possible/necessary
- incorporate operational and logical semantics
- let the user (client, analyst or designer) decide on how they want to view the models (the syntax)
- where possible, let the client construct their own requirements
- animate/execute requirements specifications as a means of rapid prototyping
- never force the client to use a vocabulary they don’t understand
- never compromise how the client structures their understanding of the problem
- don’t let the client make implementation decisions

The requirements model – needs to be validated

The model:

•acts as a *contract* between client and analyst

•improves communication by attacking risks ---

- •client misunderstands
- •client informs/communicates
- •analyst misunderstands
- •analyst misleads

•will act as *contract* with designers

"I think you misunderstood me when I said I wanted our factory to go all green."

Requirements case study : *incompleteness*

A typical example is that of a stack (or queue):

- client specifies LIFO behaviour using push and pop
- the **exception** case: popping from

empty is not specified so what to do -

  - return to client and ask them what is required
  - leave it up to the implementers to decide only if the client thinks that this is best

Note: formal methods can help identify **incompleteness**

Requirements case study : *inconsistency*

A typical example is that of a double honours student

- client specifies that student can do two <u>different</u> subjects
- client allows students to change one of their subjects

Problem: by changing one subject, a student can end up studying two subjects which are the same

Solution: make the client remove the inconsistency (don't just hide a fix away in the design/implementation)

Note: formal methods can help identify **inconsistency**

Requirements case study : ***non-(implementable/feasible)***

Try and make sure you are not asked to do something which can't be done :

•Implement a set of **inconsistent** requirements

•Implement a set of **uncomputable** requirements

•Implement a set of requirements that are **unrealistic** given today's technology

Requirements case study : *under-specification*

Under-Specification occurs when requirements are too vague

Under-specification is easy to identify as it usually corresponds to the expression of an idealistic goal, leaving the reader with no idea of how one could check whether a given system actually meets the goal, or even if such a system could exist.

An example of this is an EU e-voting requirement [standard 65]:

> *"The presentation of the voting options shall be optimised for the voter."*

Requirements case study : *over-specification*

Over-Specification occurs when requirements are too concrete

Over-specification is easy to identify as it usually manifests itself in a sentence of the form: “you must use X because X does Y”.

Clearly, a requirements document would be better saying “you must do Y”, and it could even state “and X is an alternative way of guaranteeing Y”.

Otherwise, if we had a machine that “uses Z to do Y” then this machine would be rejected even though it met its requirements.

An example of this is an EU e-voting requirement [standard 66]:

> *“Open standards shall be used to ensure that the various technical components [. . .] interoperate”*

Requirements case study : ***keeping client structure***

A typical example is that of a client who structures their understanding in terms of components with which they are familiar. For example, a client who wants:

> *a system of 2 stacks where we can push elements onto one stack and pop elements of the other. When a pop is requested, all elements on the first stack are popped off 1-by-1 and pushed onto the second stack 1-by-1.. Then, the last element is popped off. Finally, all the remaining elements are popped off the second stack and pushed on the first (again, 1-by-1)*

Problem: this is in fact a queue!

Solution 1: explain queues to the client

Solution 2: transform automatically at the first design stage

Note: here the structure of the client's understanding must be respected

Problem Based Learning : a lift

Specify the requirements of a lift/elevator without making any implementation decisions:

- say *what* not *how*
- identify and formalise the client's vocabulary
- comment on validation
- how easy is it to verify a design/implementation?

Practical Work – working in teams (or alone) - specify –he requirements of a lift/elevator system… you should need about 60 minutes …… then we'll try to evaluate *how good* they are

HINT - be careful about *ambiguity*