university of groningen

An NL-based Foundation for Increased Traceability, Transparency, and Speed in Continuous Development of Information Systems

Bert de Brock
University of Groningen
Faculty of Economics and Business
The Netherlands
E.O.de.Brock@rug.nl

Contents of the talk

- Context: Problem background (w.r.t. Requirements Engineering)
- Sketch of a development path for a single functional requirement
- (Incremental/Agile) Development of a system
- A general (linguistic) structure for a development path
- Some frequently used action verbs (related to CRUD)
- Summary

Problem background (I)

Some problems in developing information systems:

- users' language/thinking vs. developers' language/thinking ⎤ *(very) old*
- user wishes are unclear (at least initially) ⎦ *problems*

- from automating known processes to enabling entirely new business models

- (very) quickly changing circumstances ⎤ *relatively newer problems*
- therefore user wishes change ('requirements drift') ⎥ *but becoming*
- 'times to market' should be shorter and shorter ⎦ *stronger and stronger*

Consequences for the (development) project:
- not within budget ⎫ *failing the*
- not within time ⎬ *basic project*
- inadequate functionality ⎭ *requirements*

Problem background (II)

First group of problems

Users' language/thinking vs. developers' language/thinking:

- Users' language/thinking: Usually (unbounded) natural language (processes, data)
- Developers' language/thinking: Schema's, models, input/output, parameters, etc.

User wishes/requests are (initially) unclear (although the requester thinks they are clear):

- Unclear functionality (what should the system do?)
- Undefined scope (boundaries of the system?)
- Which actors are involved: which people (or user groups) and (software) systems?
- Vague basic notions (even 'flight', 'study', 'exam', 'bed'; are often homonyms)

Overview of a development path for a single FR

Straightforward path from a simple user wish to an information system:

UW	User Wish	Natural language expression
↓		
US	User Story	Natural language sentence
↓		
UC	Use Case	Sequence of natural language sentences
↓		
SSD	System Sequence Diagram	Use Case in schematic form
↓		
IM	Information Machine	Mathematical (machine) model
↓		
IS	Information System	Software

From User Wish via User Story to Use Case

UW1 Register a student

US1 As an administrator, I want to
Register a student with a given name, address, and phone number

UC1 1. The administrator (user) asks the system to
Register a student with a given name, address, and phone number
2. The system uses the next unused student number as the new student number
3. The system registers the name, address, phone number, and student number
4. The system returns the assigned student number to the user
5. The system increases the next unused student number by 1

'Stepwise clarification' / 'Stepwise specification'

A sample System Sequence Diagram

1. User → System: RegisterStudent(<name>, <address>, <phone number>)**;**
2. System → System: use the next unused student number as the new student number**;**
3. System → System: register the name, address, phone number, and student number**;**
4. System → User: "Assigned student number is " <student number>**;**
5. System → System: increase the next unused student number by 1

(Often drawn in a UML-diagram)

We can distinguish 3 types of relevant basic steps (where Actor ≠ **System**):

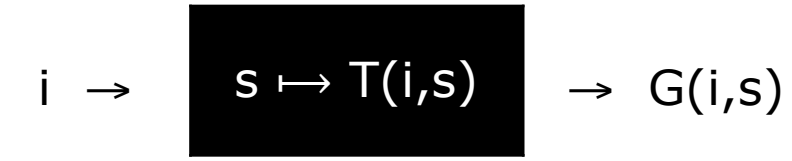
Actor → **System:**	Elucidates the inputs the system can expect	(*input step*)
System → **System:**	Elucidates the transitions (and checks) the system should make	(*internal step*)
System → Actor**:**	Elucidates the outputs the system should produce	(*output step*)

Information Machine (formal definition)

An **information machine** is a 5-tuple (I, O, S, G, T) consisting of:

- a set I (of *inputs*), called its *input space*
- a set O (of *outputs*), called its *output space*
- a set S (of *states*), called its *state space*
- a function G: $I \times S \rightarrow O$ (the *output function*),
  mapping input-state pairs to the corresponding output
- a function T: $I \times S \rightarrow S$ (the *transition function*),
  mapping input-state pairs to the corresponding next state

In a picture (with $i \in I$ and $s \in S$):



A sample Information Machine

State space Consists of

- an integer-valued component (NUSN: $\mathbb{N}$) and
- a table-valued component (STUD), with 4 attributes (+ their value sets):

  NAME: Str, ADDR: Str, PHNR: Str, SNR: $\mathbb{N}$

For this SSD:

Inputs $\{ \text{RegisterStudent}(n,a,p) \mid n \in \text{Str} \text{ and } a \in \text{Str} \text{ and } p \in \text{Str} \}$

For state s and input i = RegisterStudent(n,a,p):

Outputs G(i,s) = "Assigned student number is " & s(NUSN)

Transitions T(i,s) is the state in which the new NUSN-value is s(NUSN) + 1 and the new STUD-value is s(STUD) $\cup$ {t1};
here t1 is the function defined by
t1(SNR) = s(NUSN), t1(NAME) = n, t1(ADDR) = a, and t1(PHNR) = p

A sample Information System (Data Structure)

State space of the IM is translated to a data structure in the IS.
Sample data structure in (pseudo-)SQL:

```
CREATE DATABASE StudentRegistration        /* Student registration database
```

Data structure

```
CREATE TABLE  STUD
(NAME  VARCHAR       NOT NULL,
 ADDR  VARCHAR       NOT NULL,
 PHNR  VARCHAR       NOT NULL,
 SNR   INTEGER       NOT NULL)

CREATE VARIABLE  NUSN  AS INTEGER
```

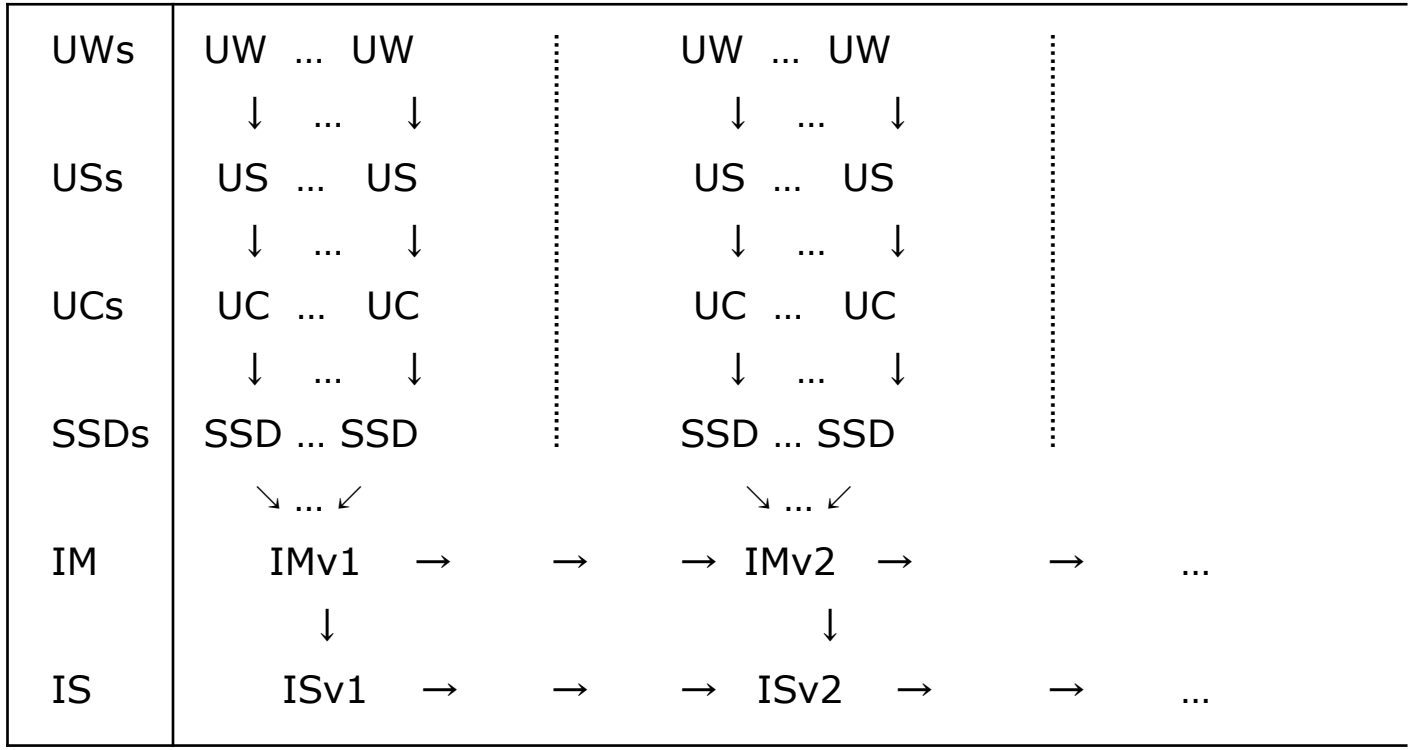
A sample procedure (implementing a user wish)

Procedure (in SQL) implementing the user wish UW1 (*Register a student*)

```
CREATE PROCEDURE RegisterStudent @name VARCHAR, @addr VARCHAR,
        @phnr VARCHAR, @output VARCHAR OUTPUT   AS

BEGIN INSERT INTO STUD (NAME, ADDR, PHNR, SNR)
                VALUES (@name, @addr, @phnr, NUSN);
      SELECT @output = "Assigned student number is " & NUSN;
      UPDATE NUSN SET NUSN = NUSN + 1
END
```

(Incremental/Agile) Development of a System



A general (linguistic) structure for a development path

Summary of the Relationship between the Subsequent Grammatical Forms

UW	α **a** β (α: action verb, β: noun phrase)
US	**As a** <role>**, I want to** α **a** β **with a given** <parameter list>
UC	First step: **The** <role> **(user) asks the system to** α **a** β **with a given** <parameter list>
SSD	First step: **User → System:** αβ**(**<parameter list>**)** where User is a <role>
IM	Inputs: αβ**(**<parameter list>**)** for all possible value combinations of <parameter list>
IS	Method/procedure αβ with <parameter list> (plus maybe an output parameter) of which the body stems from the UC/SSD-structure and the IM-details

Note the transparency and (bi-directional) traceability,
from the original user wish to the final software code and vice versa.
This will also speed up development, especially in case of changes/adaptions

Some frequently used action verbs (CRUD)

Some basic action verbs for functional requirements:
Cf. the well-known CRUD-operations (Create, Read, Update, and Delete)

CRUD	**Some alternatively used action verbs**
Create	Register, Add, Enter, Insert
Read	Retrieve, View, See, Search
Update	Refresh, Change, Modify, Edit, Alter, Adapt, Replace, Rename
Delete	Remove, Destroy

A general linguistic structure for some frequently used action verbs

Grammatical Forms for each CRUD-Case (Basic FR Category)

	Verb α	**Form in the US/UC-part**	**Form in the SSD/IM-part**
C	Create/Register/…	α **a** β **with a given** <parameter list>	αβ**(**<parameter list>**)**
R	Read/Retrieve/…	α **the** β **with a given** <ID>	αβ**(**<ID>**)**
U	Update/Refresh/…	α <par. sub-list> **of the** β **with a given** <ID>	αβ**(**<ID>, <par. sub-list>**)**
D	Delete/Remove/…	α **the** β **with a given** <ID>	αβ**(**<ID>**)**

Summary

- We sketched a straightforward development path for functional requirements,
  all the way from initial user wishes up to a software realization
  (say, a *method* in an OO-system or a (stored) *procedure* in a relational system)
- Since the path is NL-based, users can write and/or validate it (up to the UC/SSD)
- We presented some general linguistic structures and forms for such paths
- Those structures made the paths transparent and easily traceable (back and forth)
- We also memorized some frequently used action verbs (CRUD)