university of groningen

faculty of economics and business

2019002-EEF

On Incremental and Agile Development of (Information) Systems

April 2019

Bert de Brock

SOM is the research institute of the Faculty of Economics & Business at the University of Groningen. SOM has six programmes:
- Economics, Econometrics and Finance
- Global Economics & Management
- Innovation & Organization
- Marketing
- Operations Management & Operations Research
- Organizational Behaviour

Research Institute SOM
Faculty of Economics & Business
University of Groningen

Visiting address:
Nettelbosje 2
9747 AE Groningen
The Netherlands

Postal address:
P.O. Box 800
9700 AV Groningen
The Netherlands

T +31 50 363 7068/3815

www.rug.nl/feb/research

On Incremental and Agile Development of (Information) Systems

Bert de Brock
University of Groningen, Faculty of Economics and Business
e.o.de.brock@rug.nl

On Incremental and Agile Development of (Information) Systems

Bert de Brock
Faculty of Economics and Business
University of Groningen
PO Box 800, 9700 AV Groningen, The Netherlands
E.O.de.Brock@rug.nl

Abstract

Context:
Obstacles during the development of an information system (IS) are: (1) users have a (sometimes completely) different language and way of thinking than developers, (2) user wishes are often unclear (at least initially), and (3) the user wishes also change over time. Moreover, (a) the 'times to market' must be shorter and shorter and (b) the environments are changing quicker and quicker. Therefore, IS-development methods changed from 'waterfall'-like to incremental, agile and even 'continuous' over time. Understandability, f*lexibility, traceability, and development speed during the development (and evolution) of* information systems become more and more important.
Objective:
This paper aims to further improve understandability, *flexibility, traceability, and development speed during the development (and evolution) of* information systems.
Method:
The paper sketches a straightforward development path for the (incremental) development of functional requirements for (information) systems, from initial user wishes all the way to a running system. The emphasis is on understandability (by the user resp. developer), *flexibility, traceability, and development speed.*
Results:
The development path for a functional requirement proceeds from *user stories* via *use cases* and their *system sequence diagrams* to a so-called *information machine* and then to a realization (implementation), an *information system*. An evolutionary development path for a whole system is presented as well. We showed the relationship with the fundamental *ANSI-SPARC three-level architecture*, but extended from databases to general information machines. We presented a series of increments to illustrate the practical application and effects of our theory.
Conclusions:
The development path for a functional requirement enables 'stepwise clarification' and 'stepwise specification'. This improves understandability, *flexibility, traceability, and speed of development.* The paper presents a practical theory with very straightforward, transparent, traceable and incremental/agile development paths for an individual functional requirement and for a whole system, which naturally lead to modular systems.

Keywords

incremental/agile/continuous development; development path; user story; use case; system sequence diagram; information machine; property preservation; complete induction for information machines; information system; ANSI-SPARC three-level architecture

Introduction

In [1,2] we shortly introduced a straightforward development path for the (incremental) development of functional requirements for (information) systems, from initial user wishes up to a running system. In this paper we work out and illustrate that approach in more detail. The development path proceeds from *user stories* via *use cases* and their *system sequence diagrams* to a so-called *information machine* and then to a realization, an *information system*. We note that for this goal we sometimes had to 'tune' these existing notions to one another.

Up to the IM, the development should be implementation-independent. We point out that an IM could have many different realizations, e.g., by means of a human servant (say a clerk), an 'SQL servant' (i.e., a computer with SQL software), or a 'Java servant' (i.e., a computer with Java software), for instance. We will illustrate this by showing how the same IM could be realized by means of a human servant (e.g., a clerk) or by an 'SQL servant'.

We will also illustrate how an incremental or agile application of this approach can lead to modularity and transparency of the resulting system.

The paper is organized as follows: Section 1 contains an introductory example. Section 2 recalls the background notions *user story* (US), *use case* (UC) and *system sequence diagram* (SSD). Section 3 presents the notion of an *information machine* (IM): An IM can receive an input and will then produce an output and might change its state. Section 4 explains how an IM will behave when it receives a *sequence* of inputs. Section 5 introduces the notions of *property preservation* and *complete induction for information machines* as a potential means to prove additional state properties of IMs, which we illustrate with an example. Section 6 sketches the *development path* from USs via UCs and SSDs to an IM. An information machine is a *blueprint*, and can have many different *implementations*, as Section 7 points out. We call the implementation of an information machine an *information system*. Section 8 presents some extensions/increments of our running example, and sections 9, 10 and 11 then treat *incremental*, *agile*, and *continuous development* of (information) systems more generally.

Section 7 also shows the relation with the fundamental *ANSI-SPARC three-level architecture*, but now extended from databases to IMs in general: USs, UCs and SSDs belong to the *external level*, an IM belongs to the *conceptual level*, and implementations of an IM belong to the *internal level*.

Finally, the appendices illustrate a complete development path for our running example: Appendix A shows the finally resulting USs, UCs and SSDs (external level), Appendix B the finally resulting IM (conceptual level), and Appendix C an implementation using an 'SQL servant' (internal level). They also show the traceability and the modularity of the resulting system when developed in this way.

1 An introductory example

We start with a first description of our very simple running example of an information 'system':

Example 1: A very simple student administration 'system'

A fictitious university used to have a very simple student administration in its early days. In the beginning there were only a few students. The university wanted to register only the name and the student number of each of its students. This was 'implemented' as follows:

> Upon request of a university employee a servant could register the name of a new student and assign a new student number to that student. Therefore, the servant also kept track of the next unused student number (initially starting with number 1). Once in a while a student left the university, e.g. because the student finished his* studies. (*: There were no female students in those days.)
>
> Upon request of a university employee the servant could remove a student with a given student number from the administration.

Therefore, there were only two kinds of usage (or 'user stories') of this 'system':

US1: A university employee wants to: Register a student with a given name
US2: A university employee wants to: Remove a student with a given student number

These two user stories worked out in more detail (called 'use cases'):

UC1
1. A university employee (the 'actor') asks the servant to register a student with a given name
2. The servant writes down the name (with a quill pen on parchment)
3. The servant assigns the next unused student number to the new student
4. The servant returns the assigned student number to the employee
5. The servant increases the next unused number by 1 (using a slate, a sponge, and a crayon)

UC2
1. A university employee (the 'actor') asks the servant to remove a student with a given number
2. The servant strikes out the student info (with an extra thick quill pen on that parchment)
3. The servant tells the employee that he did it (or that the student number was unknown)

2 Background notions: User stories, use cases and system sequence diagrams

We recall some background notions from [1].

Informally speaking, a **user story** (US) is a 'wish' of a (future) user which the system should be able to fulfil (see also [3] for instance), e.g. the wish of a university employee to register a student. Example 1 contains two user stories, US1 and US2.

A **use case** (UC) is a text in natural language that describes the steps of a typical usage of the system (see also [4,5] for instance). Initially, use cases can be produced by (future) users of the system, domain experts or (other) staff members, or officials from the organization for which the system has to be built. Initially written use cases might need to be sharpened/enhanced/detailed in order to clarify what the system should do exactly. A (business) analyst might help to produce sharpened versions of use cases. Example 1 contains two (already sharpened) use cases, UC1 and UC2.

A **system sequence diagram** (SSD) of a use case is a 'diagram' that depicts the interaction between the primary actor (user), the system, and its supporting actors (if any), including the messages between them (see Chapter 10 of [6], for instance). An SSD is a kind of stylised UC that makes the prospective inputs, state changes, and outputs of the anticipated system more explicit. Although an SSD can be drawn as a fancy picture (see [6,7], for instance), we only denote its bare essence. Example 2 shows two SSDs.

We distinguish 3 types of relevant basic interaction steps in SSDs:

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

Example 2: Simplified system sequence diagrams for our student administration system

Example 1 contains two use cases, UC1 and UC2. We present a simplified SSD for each of the two use cases. In these use cases, the user is a university employee and the 'system' is the servant (with his belongings).

Schematically, with the numbers referring to the UC-steps (and with variables between brackets):

SSD1, for UC1
1. User → System: RegisterStudent(<name>)
2. System → System: write down the name
3. System → System: assign the next unused student number to the new student
4. User ← System: "Assigned student number: " <number>
5. System → System: increase the next unused student number by 1

SSD2, for UC2

1. User → System: RemoveStudent(<number>)
2. System → System: strike out the student info if the student (number) was known
3. User ← System: "Done" or "Unknown student number"

3 Formal modelling: Information machines

We introduce the notion of an *information machine*:

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

- a set I (of inputs)
- a set O (of outputs)
- a set S (of states)
- a function G: S x I → O, mapping pairs of a state and an input to the corresponding output
- a function T: S x I → S, mapping pairs of a state and an input to the corresponding next state

The notation f: X → Y indicates that f is a function with dom(f) = X and rng(f) ⊆ Y
(where dom(f) denotes the domain of f and rng(f) denotes the range of f).
We sometimes write f_x instead of f(x), especially when f(x) is a function again.

Intermezzo: Comparing machines

Our notion of *information machine* is equivalent to the notion of *data machine* in [8].
It is a – not necessarily finite – Mealy machine without a special start state (see [9,10]).
The table below shows for each kind of machine its ingredients and their restrictions:

Ingredient: Restriction:	**set S**	**S finite**	**$s_0 \in S$**	**set I**	**I finite**	**T: S x I → S**	**set O**	**O finite**	**G: S x I → O**
Mealy [9]	+	+	+	+	+	+	+	+	+
Pieper [8]	+			+		+	+		+
De Brock	+			+		+	+		+

We could have chosen for other (but equivalent) forms for G and T: T: I → (S → S) and G: I → (S → O)
Or we could even have chosen for only one (combined) function: F: I → (S → S x O)

In those cases, each input i ∈ I leads to a function T_i assigning a 'new' state to an 'old' state and a function G_i assigning an output to the 'old' state or, in the second case, to a function F_i assigning a 'new' state and an output to an 'old' state.

If the system also communicates with supporting actors, say other systems, then we still distinguish three types of relevant basic interaction steps, but with (primary) 'User' generalized to 'Actor', where an actor can be any other system. Expressed in terms of an information machine (IM):

Actor → System: Elucidates the needed set I of inputs of the IM
System → Actor: Elucidates the needed set O of outputs and output function G of the IM
System → System: Elucidates the needed set S of states and transition function T of the IM

We informally describe an *information machine* for our simple student administration from examples 1 and 2. Note that the ingredients follow directly and quite naturally from the SSDs!

Example 3: An information machine for our simple student administration thus far

Below we consecutively introduce the *inputs*, the *outputs*, the *states*, the *output function* and the *transition function* for our information machine.

Schematically, the possible **inputs** and (corresponding) **outputs** are (with variables between brackets):

Input	Output
RegisterStudent(<name>)	"Assigned student number: " <number>
RemoveStudent(<number>)	"Done" or "Unknown student number", depending on the presence resp. absence of the number in the administration

A **state** essentially consists of 2 components: a table of students (with their name and student number) and a number, i.e., the next unused student number. As an example:

Next unused student number: 5

Students:

Name	Number
J. Smith	1
A. Adams	2
J. Brown	3
[redacted]	[redacted]

In this example, student no. 4 already left the university again.

The **output function** maps a pair of a state and an input to the corresponding output as follows (where the output might depend on the presence or absence of the number in the student table):

State	Input	Output	Condition	Related SSD + steps
s	RegisterStudent(x)	"Assigned student number: " s(NN)		SSD1: 1, 4
s	RemoveStudent(n)	"Done"	if $n \in s(ST) \,\mathbb{T}$ Number	SSD2: 1, 3
		"Unknown student number"	if $n \notin s(ST) \,\mathbb{T}$ Number	SSD2: 1, 3

where s(NN) denotes the next unused student number in state s, s(ST) the student table in state s, and $s(ST) \,\mathbb{T}$ Number the set of all values in the Number column of the table s(ST), where we define for a table T and an attribute a of T: $\underline{T \,\mathbb{T}\, a} = \{ t(a) \mid t \in T \}$, i.e., the set of all a-values in table T

In the state above:
- the input RegisterStudent(A. Adams) would lead to the output "Assigned student number: 5",
- the input RemoveStudent(1) would lead to the output "Done", and
- the input RemoveStudent(4) would lead to the output "Unknown student number".

The **transition function** maps a pair of a state and an input to the corresponding next state as follows:

State	Input	Next state Next unused student number	Next state Table	Related SSD plus steps
s	RegisterStudent(x)	s(NN) + 1	$s(ST) \cup \{ [x; s(NN)] \}$	SSD1: 2, 3, 5
s	RemoveStudent(n)	s(NN)	$\{ t \in s(ST) \mid t(Number) \neq n \}$	SSD2: 2

where in this case $\{ [x; s(NN)] \}$ denotes the single row with student name x and student number s(NN), and $\{ t \in s(ST) \mid t(Number) \neq n \}$ denotes the table s(ST) minus 'all' rows with student number n. (Actually, in Example 5 we will prove that there is at most one row with student number n.)

4 Sequences of inputs and corresponding outputs

If an information machine (IM) receives a sequence of inputs, then the IM goes through a sequence of states and produces a sequence of outputs. We illustrate this in the next example.

Example 4: A sequence of inputs and corresponding outputs

If our sample IM is in the state shown in Example 3 and receives the three consecutive inputs

RegisterStudent(A. Adams) ; RemoveStudent(1) ; RemoveStudent(4)

then the output sequence would be

"Assigned student number: 5" ; "Done" ; "Unknown student number"

and the new state would be:

Next unused student number: 6

Students:

Name	Number
[redacted]	[redacted]
A. Adams	2
J. Brown	3
[redacted]	[redacted]
A. Adams	5

In general, each next state is the result of applying the transition function to the combination of the previous state and the received input, and each next output is the result of applying the output function to that same combination of the previous state and the received input.

Formally: If an IM (I, O, S, G, T) receives a sequence $<i_1; i_2; \ldots; i_n>$ of inputs and initially is in state s_0, then the IM goes through the sequence $<s_1; s_2; \ldots; s_n>$ of states and produces the sequence $<o_1; o_2; \ldots; o_n>$ of outputs where, for all k from 1 up to n, state s_k is defined as $T(s_{k-1}, i_k)$ and output o_k is defined as $G(s_{k-1}, i_k)$.

5 Property preservation and complete induction for information machines

If a property holding for a state also holds for any next state in an information machine, then we say that that information machine *preserves* that state property. Defined formally:

An inf. machine (I, O, S, G, T) **preserves** property P $\Leftrightarrow$ P(s) implies P(T(s,i)) for all s $\in$ S and i $\in$ I

Complete induction for information machines

If an information machine preserves a property P and starts in a state s_0 that has that property, then it is clear that that information machine will always be in a state with that property.

Example 5: Property preservation and complete induction for our sample information machine

We will prove that our sample information machine preserves the following pair of properties:
(P1) the next unused student number is larger than each student number in the student table, and
(P2) each student number in the student table is unique

P1(s): s(NN) > k for all k $\in$ s(ST) $\mathbb{T}$ Number
P2(s): The attribute Number is u.i. (uniquely identifying) in the table s(ST)

Note: For a table T and an attribute a of T we define:
a is uniquely identifying in T $\Leftrightarrow$ t $\neq$ t' implies t(a) $\neq$ t'(a) for all t $\in$ T and t' $\in$ T

We can prove the preservation of this property pair by using the specification of the transition function of our information machine, given at the end of Example 3:

State	Input	Next state Next unused student number	Next state Table
s	RegisterStudent(x)	s(NN) + 1	s(ST) $\cup$ { [x; s(NN)] }
s	RemoveStudent(n)	s(NN)	{ t $\in$ s(ST) | t(Number) $\neq$ n }

The proof for P1 runs as follows:

- if i = RegisterStudent(x) then P1(s) – i.e. s(NN) > k for all k ∈ s(ST) 𝕋 Number – clearly implies P1(T(s,i)), i.e. s(NN) + 1 > k for all k ∈ s(ST) 𝕋 Number ∪ { s(NN) }
- if i = RemoveStudent(n) then P1(s) – i.e. s(NN) > k for all k ∈ s(ST) 𝕋 Number – clearly implies P1(T(s,i)), i.e. s(NN) > k for all k ∈ { t ∈ s(ST) | t(Number) ≠ n } 𝕋 Number

The proof for P2 now runs as follows (where we need P1 to prove P2 too):

- if i = RegisterStudent(x) then P2(s) – i.e. Number is u.i. in s(ST) – and P1(s) – i.e. s(NN) > k for all k ∈ s(ST) 𝕋 Number – imply P2(T(s,i)), i.e. Number is u.i. in s(ST) ∪ { [x; s(NN)] }
- if i = RemoveStudent(n) then P2(s) – i.e. Number is u.i. in s(ST) – clearly implies P2(T(s,i)), i.e. Number is u.i. in { t ∈ s(ST) | t(Number) ≠ n }

Note that the proof is modular ('incremental') w.r.t. the possible inputs.
The following state has the properties P1 and P2 (because s(ST), the student table, is empty):

Next unused student number: 1

Students:

Name	**Number**

So, if our information machine starts in this 'empty' state, then our information machine will always be in a state with properties P1 and P2 (complete induction for our sample information machine).

6 From user stories via use cases and system sequence diagrams to an IM

Starting from the 2 USs and the UCs (in Example 1) via their corresponding SSDs (in Example 2) we were able to define our IM (in Example 3). In a scheme (where the arrows indicate what is input for what):

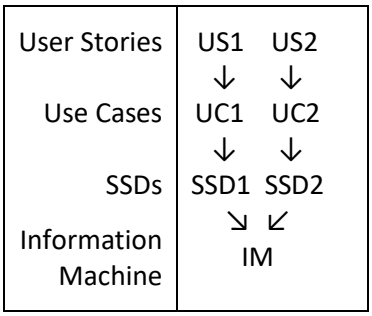
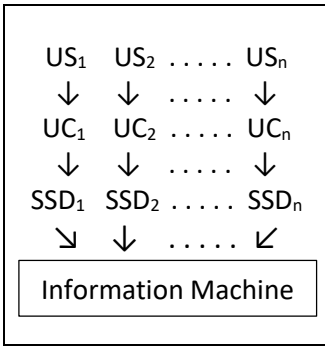


Figure 1: From USs to an IM

We can generalize this scheme as follows (where n could be large):



Short texts in natural language, each describing a 'wish' of a (future) user which the system should be able to fulfil

Texts in natural language, each describing the steps of a typical usage of the system

Diagrams, each depicting for one UC the interactions between the user, the system, and its supporting actors, including the messages between them

Formal/Conceptual model of the system, including the messages between the system and its environment

Figure 2: The relation between USs, UCs, SSDs, and an IM

7 Realizations/Implementations of an information machine

Information machines can be considered as *blueprints*. An information machine can have many different kinds of realizations/implementations. For instance, an information machine can be realized by a human servant (such as in our running example), by an 'SQL servant' (i.e., a computer with SQL software, as we will describe in Appendix C), or by a 'OO servant' (i.e., a computer with OO software).

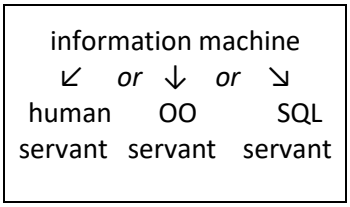


Figure 3: Different kinds of realizations of an IM

If we combine this picture with the previous ones then we can indicate the relation with the fundamental ANSI-SPARC three-level architecture, see e.g. [11,12], but now extended from databases to information machines in general:

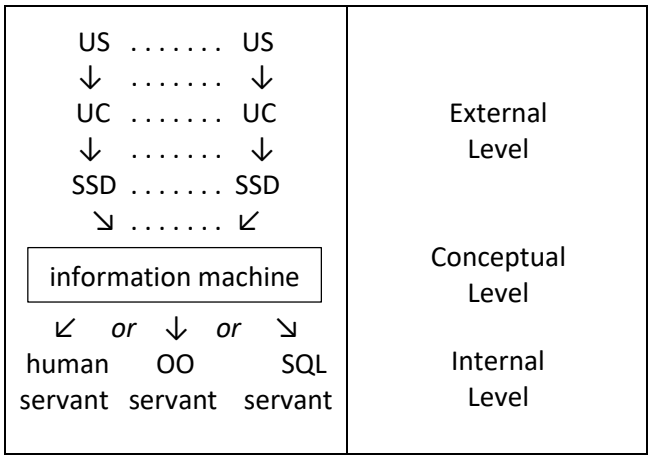


Figure 4: Relation with the ANSI-SPARC three-level architecture

In words: USs, UCs and SSDs belong to the **external level**, an IM belongs to the **conceptual level**, and any realization belongs to the **internal level**.

So far we already described a realization of our sample information machine by a human servant (who used a parchment roll and quill pen as well as a slate, sponge, and crayon). In Appendix C we will describe a realization of the same information machine by means of an 'SQL servant'.

We are inclined to call the implementation of an information machine an **information system** (IS). According to the literature an information system has a Boundary, Users, Processors, Storage, Inputs, Outputs and Communication networks (see [13,14]).

This all applies to our running example, where the university initially only wanted to register the name and the student number of each of its students, which determined the initial system boundary. Furthermore, initially the users were some (authorized) university employees only, the processor is the servant with his quill pen, sponge, and crayon, the storage consists of the parchment roll and the slate (which is rewritable), the inputs and outputs were gradually defined in our examples, and the servant's box office was the only communication network.

8 Examples of incremental and agile development of an information system

Information systems in practice are really sophisticated, i.e. supporting a lot of use cases, resulting in very large input sets, output sets, state sets, and with complicated output functions and transition functions. Moreover, in practice such systems are often under continuous development ('under construction'), just as a city for instance.

Instead of defining and developing such a sophisticated information system in one go ('big bang'), including 'all' functionality that is needed – as might be suggested in Section 6 – such systems are often defined and developed incrementally, i.e., starting with a simple, small version (as we did) and extending/adapting it in several small steps into larger, more sophisticated versions.

In the next 3 examples we present some subsequent adaptions of our sample information system. In terms of the four basic functions known in the literature as CRUD (Create, Read, Update and Delete; see [15,16] for instance), we already showed a Create (US1) and a Delete (US2). Example 6 shows a Read and Example 8 shows an Update. Example 7 shows several changes in the structure.

Example 6: Extending our sample 'system' (with a Read)

After a while it turned out that university employees sometimes needed to know the student number of a student. Therefore, another usage of the system emerged. Formulated as a user story:

US3: A university employee wants to Retrieve the student number of a student with a given name

Worked out in more detail, in a use case:

UC3
1. A university employee asks the servant for the student number of a student with a given name
2. The servant searches for all students with that name (maybe students have the same name)
3. The servant returns the corresponding student numbers to the employee

SSD3, for UC3
1. User→ System: RetrieveNumber(<name>)
2. System→ System: search for all students with that name
3. User ← System: <set of corresponding student numbers>

Now we consecutively introduce the additional *inputs* and *outputs*, the *state space* (i.e., the set of states), the extension of the *output function* and the extension of the *transition function* for our IM. The additional **inputs** and (corresponding) **outputs** of our IM are:

Input	**Output**
RetrieveNumber(<name>)	<set of corresponding student numbers>

The **state space** of our IM remains the same. The **output function** of our IM is extended as follows:

State	**Input**	**Output**
s	RetrieveNumber(x)	{ t(Number) | t ϵ s(ST) and t(Name) = x }

where { t(Number) | t ϵ s(ST) and t(Name) = x } denotes the set of student numbers of all students in the student table s(ST) with name x.
In this UC, the state always remains the same, so our **transition function** is simply extended by

State	**Input**	**Next state**
s	RetrieveNumber(x)	s

Consequently, our IM preserves all its state properties with this trivial extension. Note that the extension is 'incremental'/modular: We did not need to change any existing part of the IM.
In the state shown in Example 4, the input RetrieveNumber(A. Adams) would essentially result in the output "2, 5".

The next (innocent looking) adaption causes several changes: small ones in two US/UC/SSD-triples and several in the IM.

Example 7: Another adaption of our sample 'system' (changing several existing parts of our IM)

After a few centuries the first female student was allowed to come in (and actually did come in). Therefore, the university wanted to keep track of the gender of the students as well.

This led to (small) changes in the US1/UC1/SSD1-triple and in the US3/UC3/SSD3-triple. The changed user stories are given below (changes are underlined). The subsequent changes in the UCs and SSDs are straightforward then. (They can also be found in Appendix A.)

US1$^+$: A university employee wants to Register a student with a given name and gender
US3$^+$: A university employee wants to Retrieve the student info of a student with a given name

It also led to several changes of the information machine:

- The student table got an extra column, Gender, with 2 possible values: "M" and "F", changing the structure of states and hence the **state space** of our IM
- All existing rows in the student table got the value "M" for Gender, changing the old **current state** to the new current state
- The input RegisterStudent(<name>) changed into RegisterStudent(<name>, <gender>) and the input RetrieveNumber(<name>) changed into RetrieveStudent(<name>), because the student information was not limited to Number anymore (but included the gender as well). As a consequence, some of the **inputs** and (corresponding) **outputs** of our IM changed:

Input	**Output**
RegisterStudent(<name>, <gender>)	"Assigned student number: " <number>
RetrieveStudent(<name>)	<student info of all students with that name>

- The **output function** of our IM changed as follows:

State	**Input**	**Output**
s	RegisterStudent(x, y)	"Assigned student number: " s(NN)
s	RetrieveStudent(x)	{ t ∈ s(ST) | t(Name) = x }

- The **transition function** of our IM changed as follows:

State	**Input**	**Next state** Next unused student number	**Next state** Table
s	RegisterStudent(x, y)	s(NN) + 1	s(ST) ∪ { [x; s(NN); y] }
s	RetrieveStudent(x)	s(NN)	s(ST)

where in this case { [x; s(NN); y] } denotes the single row with student name x, student number s(NN), and gender y.

Directly after the first female student was registered, with say input RegisterStudent(A. Jacobs, F) and output "Assigned student number: 12345", the state had the following structure:

Next unused student number: 12346

Students:

Name	**Number**	**Gender**
. . .	. . .	M
:	:	:
:	:	:
:	:	:
. . .	. . .	M
A. Jacobs	12345	F

Our servant 'implemented' all this by writing an 'M' or an 'F' immediately after each student number on the parchment roll.

Note that after all these changes the state still *has* the properties P1 and P2 and our changed IM still *preserves* those properties. Phrased informally, these properties were:

P1: The next unused student number is larger than each student number in the student table
P2: Each student number in the student table is unique

Making changes to the IM itself can be considered as the usage of another IM, namely an information machine that produces information machines. We will come back to this in a later paper.

Example 8: Extending our sample 'system' again (with an Update)

After a while it turned out that female students sometimes changed their name (because they got married). This was something new (…) and led to a new user story, initially formulated as follows:

US4f: Change the name of a female student with a given student number

Shortly after it was realized that such a user story might be useful for male students too, leading to an adapted, more general user story (simply by deleting the restriction "female"):

US4: Change the name of a ~~female~~ student with a given student number

This leads to an adaption of the use case for user story US4f, with the following result (where deleted text is struck out and new text is underlined):

UC4
1. A ~~female~~ student asks the servant to register her/his new name, showing the student number
2. The servant changes ~~her~~ the name into the new name (after checking a ~~(marriage)~~ certificate)
3. The servant tells the student that he did it (or that the student number was unknown)

The extension from only female to all students did not really influence the corresponding SSD:

SSD4, for UC4
1. User → System: ChangeName~~Female~~Student(<number>, <new name>)
2. System → System: change the old name into the new name if the student number was known
3. User ← System: "Done" or "Unknown number"

Clearly the additional **inputs** and (corresponding) **outputs** of our IM are:

Input	**Output**
ChangeNameStudent(<number>, <new name>)	"Done" or "Unknown number"

The **state space** of our IM remains the same.
The **output function** of our IM is extended as follows (distinguishing two situations):

State	**Input**	**Output**	**Condition**
s	ChangeNameStudent(n, x)	"Done" "Unknown number"	if $n \in s(ST) \rangle\!\rangle$ Number if $n \notin s(ST) \rangle\!\rangle$ Number

The **transition function** of our IM is extended as follows:

State	**Input**	**Next state** Next unused number	**Next state** Table
s	ChangeNameStudent(n, x)	s(NN)	$\{ t \mid t \in s(ST) \text{ and } t(Number) \neq n \} \cup$ $\{ [x; n; t(Gender)] \mid t \in s(ST) \text{ and } t(Number) = n \}$

where in this case [a; b; c] denotes the row with student name a, student number b, and gender c.

Our servant 'implemented' this by striking out the old name (with an extra thick quill pen with black ink) and writing the new name there with white ink.

Note that after this change the state still *has* the properties P1 and P2 and our extended IM still *preserves* those properties. Also note that the extension is 'incremental'/modular again: It did not change any existing part of the IM.

We recall that the 4 basic functions known in the literature as CRUD (Create, Read, Update and Delete) are now all represented by our user stories:

Name	Alternatively used names	Examples: Our (latest) user stories
Create	Register, Add, Enter	US1^{+}: Register a student with a given name and gender
Read	Retrieve, View, Show, Search	US3^{+}: Retrieve the student info of a student with a given name
Update	Change, Modify, Edit, Alter	US4 : Change the name of a student with a given student number
Delete	Remove, Destroy, Deactivate	US2 : Remove a student with a given student number

9 Incremental development of (information) systems

After this series of subsequent extensions and adaptions of our sample information system we will treat incremental development of (information) systems more generally.

The next picture indicates which examples introduced which user stories, which use cases, which SSDs and which versions of our IM and IS, and the arrows indicate what constituted input for what.

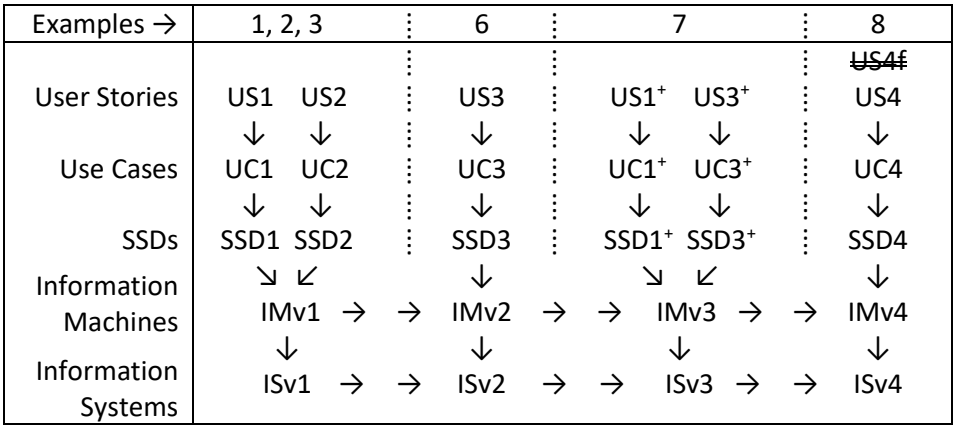


Figure 5: Incremental development path of our sample information system

It already indicates some structure in our small scale 'incremental development':
Via one or 2 USs, UCs and their corresponding SSDs (and the previous IM-version) we defined an initial (resp. next) version of our IM, and based on the IM (and the previous IS-version) we defined an initial (resp. next) version of our IS. This can be generalized easily:

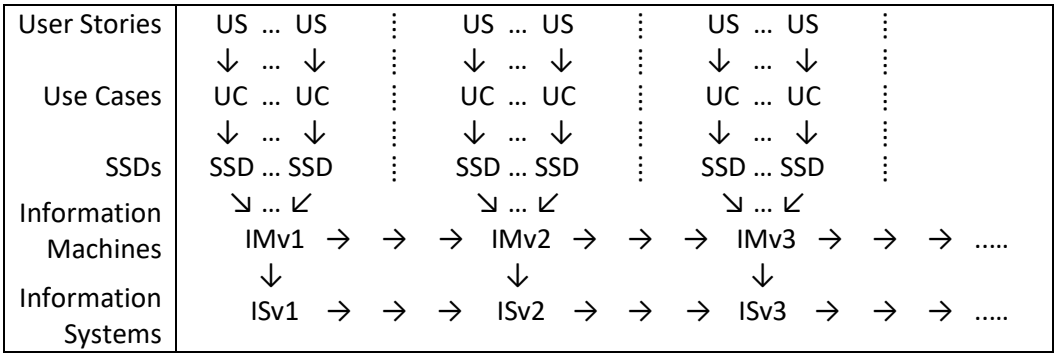


Figure 6: Incremental development path of an information system

So, via one or more USs, UCs, and their corresponding SSDs (and the previous version of the IM) we can define an initial (resp. next) version of the IM, and based on the IM (and the previous IS-version) we can define an initial (resp. next) version of the IS.

The order in which the user stories are developed could depend on story size, business value, possible precedence relations, and uncertainty in velocity prediction, for instance [17].

10 Agile development

One cycle might contain only a few USs, UCs and their corresponding SSDs, maybe even only *one* US, UC and SSD. Or maybe even *less than* one full UC: In a more agile development process a simple 'core' scenario (or 'main success scenario') of a (yet unclear) 'full' UC might be delivered first, followed by 'fuller' versions in subsequent cycles (e.g., see [6]). So, existing US/UC/SSD-triples might also be adapted, like in Example 7. Short cycles especially hold in case of *daily/nightly builds* (see [18]) and *continuous integration* (see [19,20]).

11 Continuous development of (information) systems

Note that such an incremental development can go on 'forever'. In a sense, this development process is cyclic and can be (almost) continuous. So, the following picture might be more appropriate:

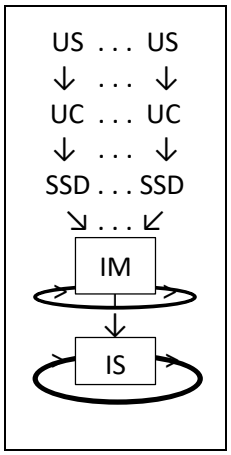


Figure 7: Continuous development path of an information system

The picture tries to express that via one or more USs, UCs, and their corresponding SSDs (and the previous version of the IM) we can define an initial (resp. next) version of the IM, and based on the IM (and the previous IS-version) we can define an initial (resp. next) version of the IS.

Results

In this paper we incrementally presented a theory on *Incremental development of (information) systems*, starting with simple ('minimal') examples and simple notions making the basics clear, and then gradually introducing more complicated notions and examples.

We placed the notions *user story*, *use case*, and *system sequence diagram* in line and linked them to the notion of an *information machine*: The set of SSDs of an application determine the *inputs*, the *outputs*, and the *output function* of the IM. We depicted a *development path* from USs via UCs and SSDs to an IM. We also defined the notions of *property preservation* and *complete induction for information machines*, as a potential means to prove additional state properties of IMs, which we illustrated with an example.

After presenting several extensions of our sample information machine, including examples of all four *CRUD-functions* and of structural changes of our information machine, we treated *incremental*, *agile*, and *continuous development* of (information) systems more generally.

We pointed out that an IM is a *blueprint*, and we emphasized that it can have completely different *implementations*. We call the implementation of an IM an *information system*. In our examples in the appendices we show how the user stories directly lead to the input sets of the IM and, when the IM is implemented in, e.g., SQL how those input sets in turn directly correspond to (stored) procedures.

We also showed the relation with the fundamental *ANSI-SPARC three-level architecture* but extended from databases to information machines in general: USs, UCs and SSDs belong to the external level, an IM belongs to the *conceptual level*, and implementations of an IM belong to the *internal level*.

Finally, the appendices illustrate a complete development path for our running example They also show the traceability and the modularity of the resulting system when developed in this way.

Conclusion

The paper works out a practical theory with a very straightforward, transparent, traceable and incremental/agile development path: From *user stories* via *use cases* and their corresponding *system sequence diagrams* to an *information machine*, and then to a realization, an *information system*. Our approach links these notions together by crossing the boundaries of several (sub)disciplines, such as requirements engineering, machine theory, and (database) systems development. Developing information systems in this manner can naturally lead to modular systems.

Another contribution of the paper is the formal definition of the notions of *property preservation* and *complete induction for information machines*, as a potential means to prove additional state properties of information machines.

Current and future work

Situations in practice can be much more complicated. Therefore, we are currently extending our theory with additional, extended, and/or more complicated issues, such as development patterns, advanced grammars for SSDs, complicated SSDs, advanced notions and further terminology related to information machines, and generalization and formalization of CRUD-functions.

In near future we will also work on interacting systems, compound transactions, dynamic constraints (i.e., constraints on state *transitions*) and the relation with work flow.

References

[1] E.O. de Brock: Towards a Theory about Continuous Requirements Engineering for Information Systems, CRE: 4th Workshop on Continuous Requirements Engineering, REFSQ, 2018

[2] E.O. de Brock: Towards a theory about Incremental development of information systems (extended abstract), ICT.OPEN, 2018

[3] G.G. Lucassen: Understanding User Stories, PhD thesis, Utrecht University, 2017

[4] I. Jacobson et al: Use Case 2.0: The Guide to Succeeding with Use Cases, Ivar Jacobson Int., 2011

[5] https://en.wikipedia.org/wiki/Use_case

[6] C. Larman: Applying UML and patterns, Addison Wesley Professional, 2004

[7] https://en.wikipedia.org/wiki/System_sequence_diagram

[8] F.T.A.M. Pieper: Data machines and interfaces, PhD thesis, TU Eindhoven, 1989

[9] G.H. Mealy: A Method for Synthesizing Sequential Circuits, Bell System Technical J., 1045–1079, 1955

[10] https://en.wikipedia.org/wiki/Mealy_machine

[11] ANSI/X3/SPARC Study Group on DBMS: Interim Report, ACM SIGMOD bulletin, vol. 7.2, 1975

[12] https://en.wikipedia.org/wiki/ANSI-SPARC_Architecture

[13] L. Jessup and J. Valacich: Information Systems Today, Pearson, 2008
[14] https://en.wikipedia.org/wiki/Information_system
[15] J. Martin: Managing the Data Base Environment, Prentice Hall, 1983
[16] https://en.wikipedia.org/wiki/Create,_read,_update_and_delete
[17] G. van Valkenhoef et al: Quantitative release planning in extreme programming, Information and Software Technology, vol. 53.11, 1227-1235, 2011
[18] https://en.wikipedia.org/wiki/Daily_build
[19] G. Booch: Object-oriented analysis and design with applications, Addison Wesley, 1998
[20] https://en.wikipedia.org/wiki/Continuous_integration
[21] E.O. de Brock: Foundations of Semantic Databases, Prentice Hall, 1995

All links were last accessed on 2018/10/26

Appendix A: Our sample USs, UCs and SSDs

Our latest sample user stories, use cases, and system sequence diagrams

Now we bring together (the latest versions of) our sample USs, UCs, and their corresponding SSDs. There were four 'final' user stories for our 'system', together representing the four basic CRUD-functions, Create, Read, Update and Delete (see [15,16] for instance):

US1+: Register a student with a given name and gender /* Create
US2 : Remove a student with a given student number /* Delete
US3+: Retrieve the student ~~number~~ info of a student with a given name /* Read
US4 : Change the name of a ~~female~~ student with a given student number /* Update

Below, each UC is given, followed by its SSD. Schematically, with the step numbers in the SSDs referring to the corresponding step numbers in the UC (and with the variables between pointy brackets):

UC1+
1. A university employee asks the servant to register a student with a given name and gender
2. The servant writes down the name and gender (with a quill pen on parchment)
3. The servant assigns the next unused student number to the new student
4. The servant returns the assigned student number to the employee
5. The servant increases the next unused student number by 1 (using a slate, sponge, and crayon)

SSD for UC1+
1. User → System: RegisterStudent(<name>, <gender>)
2. System → System: write down the name and gender
3. System → System: assign the next unused student number to the new student
4. User ← System: "Assigned student number: " <number>
5. System → System: increase the next unused student number by 1

UC2
1. A university employee asks the servant to remove a student with a given number
2. The servant strikes out the student info (with an extra thick quill pen on that parchment), if known
3. The servant tells the employee that he did it (or that the student number was unknown)

SSD for UC2
1. User → System: RemoveStudent(<number>)
2. System → System: strike out the student info if the student (number) was known
3. User ← System: "Done" or "Unknown student number"

UC3+
1. A university employee asks the servant for the student info of a student with a given name
2. The servant searches for all students with that name (several students may have the same name)
3. The servant returns the student info of all students with that name to the employee

SSD for UC3+
1. User → System: Retrieve~~Number~~Student(<name>)
2. System → System: search for all students with that name
3. User ← System: <student info of all students with that name>

UC4
1. A student personally asks the servant to register the new name, showing her/his student number
2. The servant changes ~~her~~ the old name into the new name (after checking a ~~(marriage)~~ certificate)
3. The servant tells the student that he did it (or that the student number was unknown)

SSD for UC4
1. User → System: ChangeName~~Female~~Student(<number>, <new name>)
2. System → System: change the old name into the new name if the student number was known
3. User ← System: "Done" or "Unknown number"

Appendix B: Our sample information machine

Our latest sample information machine

We consecutively specify the *states*, the *inputs*, the *outputs*, the *output function* and the *transition function* for our latest sample information machine:

Each **state** consists of 2 components: a table of students (with their name, student number, and gender) and a number, namely the next unused student number.

Schematically, the possible **inputs** and (corresponding) **outputs** are (variables between brackets):

Input	Output
RegisterStudent(<name>, <gender>)	"Assigned student number: " <number>
RetrieveStudent(<name>)	<student info of all students with that name>
ChangeNameStudent(<number>, <new name>)	"Done" or "Unknown number"
RemoveStudent(<number>)	"Done" or "Unknown student number"

The **output function** in a schema (sometimes distinguishing two situations):

State	Input	Output	Condition
s	RegisterStudent(x, y)	"Assigned student number: " s(NN)	
s	RetrieveStudent(x)	{ t ∈ s(ST) | t(Name) = x }	
s	ChangeNameStudent(n, x)	"Done"	if n ∈ s(ST) 𝕀 Number
		"Unknown number"	if n ∉ s(ST) 𝕀 Number
s	RemoveStudent(n)	"Done"	if n ∈ s(ST) 𝕀 Number
		"Unknown student number"	if n ∉ s(ST) 𝕀 Number

where s(NN) denotes the next unused student number in state s, s(ST) the student table in state s, and s(ST) 𝕀 Number the set of all values in the Number column of the table s(ST).

The **transition function** in a schema (where we must specify both components of the next state):

State	Input	Next state Next unused student number	Next state Table
s	RegisterStudent(x, y)	s(NN) + 1	s(ST) ⋃ { [x; s(NN); y] }
s	RetrieveStudent(x)	s(NN)	s(ST)
s	ChangeNameStudent(n, x)	s(NN)	{ t | t ∈ s(ST) and t(Number) ≠ n } ⋃ { [x; n; t(Gender)] | t ∈ s(ST) and t(Number) = n }
s	RemoveStudent(n)	s(NN)	{ t ∈ s(ST) | t(Number) ≠ n }

where in these cases [a; b; c] denotes the row with student name a, number b, and gender c.

Appendix C: An SQL-realization of our sample information machine

An SQL-realization of our sample information machine

We will now describe a realization of our sample IM by means of an 'SQL servant' (i.e., a computer with SQL software). The realization follows directly from the description given in Appendix B.

We start with a realization of our state space (along the lines of Chapter 9 of [21]). Below we first introduce the so-called *database name* StudentRegistration. We consider StudentRegistration as a variable that has our state space as its set of possible values. StudentRegistration has 2 components: (1) a table ST, with attributes Name, Number, and Gender, and (2) an integer variable NN.

Our sample IM up to now preserves the property that each student number in the student table is unique (property P2 in Example 5), thanks to the limited set of applications (possible inputs). Something similar holds for the property that the Gender value is always in the set {'M', 'F'}. However, we want the system to guard these intended constraints independent of the set of applications, because any new application might spoil these properties. Therefore, we explicitly require those constraints in our SQL declaration below. First, we explicitly give the set {'M', 'F'} a name by creating a domain (i.e., data type) called GenSet. Then we require that the Gender-values must come from this set. Moreover, we require the attribute Number to be unique. (The details of the SQL syntax might vary among different 'SQL servants', so locally you might need a slightly different syntax. We note that the CREATE VARIABLE syntax below is a kind of 'pseudo-SQL'.)

At the end of each line we indicate the origin of that ingredient (almost always a user story). This also makes clear which ingredient came in in which development round.

```
CREATE DATABASE StudentRegistration      /* original wish to have a student registration system

CREATE DOMAIN Genset AS CHAR(1)          /* US1+
CHECK ( @VALUE IN ('M','F') )            /* US1+

CREATE TABLE  ST                         /* US1
( Name     VARCHAR     NOT NULL,         /* US1
  Number   INTEGER     NOT NULL,         /* US1
  Gender   GenSet      NOT NULL,         /* US1+
  UNIQUE( Number ) )                     /* Appendix C

CREATE VARIABLE  NN  AS INTEGER          /* US1
```

As noted in Section 3 we can combine the output function and the transition function into 1 function:

F: I → (S → S x O): each input leads to a function assigning to an 'old' state a new state and an output

We can implement that function incrementally, using 4 (stored) procedures, one for each of the (final) USs we subsequently introduced. Each procedure has 1 or 2 input parameters and usually also an explicit output parameter. (Parameter names are preceded by an "@".)

In front of each CREATE PROCEDURE we indicate the origin of that procedure (i.e. a user story). This makes clear which ingredient came in in which development round. It clearly shows the modularity of the resulting system we developed in this way.

The correspondence between the CRUD verbs, the verbs we used, and the SQL statements is:

CRUD name	Name we used in our examples	SQL
Create	Register	INSERT
Read	Retrieve	SELECT
Update	Change	UPDATE
Delete	Remove	DELETE

```
/* US1+ */      CREATE PROCEDURE RegisterStudent @name VARCHAR, @gender GenSet,
                                    @output VARCHAR OUTPUT   AS
                BEGIN INSERT INTO ST (Name, Number, Gender) VALUES (@name, NN, @gender);
                      SELECT @output = "Assigned student number: " + NN;
                      UPDATE NN SET NN = NN + 1
                END

/* US2 */       CREATE PROCEDURE RemoveStudent @number INTEGER,
                                    @output VARCHAR OUTPUT   AS
                IF @number IN (SELECT NR FROM ST)
                THEN  DELETE FROM ST t WHERE t.Number = @number;
                      SELECT @output = "Done"
                ELSE  SELECT @output = "Unknown student number"

/* US3+ */      CREATE PROCEDURE RetrieveStudent @name VARCHAR   AS
                SELECT ~~Number~~ * FROM ST t WHERE t.Name = @name

/* US4 */       CREATE PROCEDURE ChangeNameStudent @number INTEGER, @new_n VARCHAR,
                                    @output VARCHAR OUTPUT   AS
                IF @number IN (SELECT Number FROM ST)
                THEN  UPDATE ST t SET t.Name = @new_n WHERE t.Number = @number;
                      SELECT @output = "Done"
                ELSE  SELECT @output = "Unknown number"
```

Declarations of interest: none. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

List of research reports

14001-OPERA: Germs, R. and N.D. van Foreest, Optimal control of production-inventory systems with constant and compound poisson demand

14002-EEF: Bao, T. and J. Duffy, Adaptive vs. eductive learning: Theory and evidence

14003-OPERA: Syntetos, A.A. and R.H. Teunter, On the calculation of safety stocks

14004-EEF: Bouwmeester, M.C., J. Oosterhaven and J.M. Rueda-Cantuche, Measuring the EU value added embodied in EU foreign exports by consolidating 27 national supply and use tables for 2000-2007

14005-OPERA: Prak, D.R.J., R.H. Teunter and J. Riezebos, Periodic review and continuous ordering

14006-EEF: Reijnders, L.S.M., The college gender gap reversal: Insights from a life-cycle perspective

14007-EEF: Reijnders, L.S.M., Child care subsidies with endogenous education and fertility

14008-EEF: Otter, P.W., J.P.A.M. Jacobs and A.H.J. den Reijer, A criterion for the number of factors in a data-rich environment

14009-EEF: Mierau, J.O. and E. Suari Andreu, Fiscal rules and government size in the European Union

14010-EEF: Dijkstra, P.T., M.A. Haan and M. Mulder, Industry structure and collusion with uniform yardstick competition: theory and experiments

14011-EEF: Huizingh, E. and M. Mulder, Effectiveness of regulatory interventions on firm behavior: a randomized field experiment with e-commerce firms

14012-GEM: Bressand, A., Proving the old spell wrong: New African hydrocarbon producers and the 'resource curse'

14013-EEF: Dijkstra P.T., Price leadership and unequal market sharing: Collusion in experimental markets

14014-EEF: Angelini, V., M. Bertoni, and L. Corazzini, Unpacking the determinants of life satisfaction: A survey experiment

14015-EEF: Heijdra, B.J., J.O. Mierau, and T. Trimborn, Stimulating annuity markets

14016-GEM: Bezemer, D., M. Grydaki, and L. Zhang, Is financial development bad for growth?

14017-EEF: De Cao, E. and C. Lutz, Sensitive survey questions: measuring attitudes regarding female circumcision through a list experiment

14018-EEF: De Cao, E., The height production function from birth to maturity

14019-EEF: Allers, M.A. and J.B. Geertsema, The effects of local government amalgamation on public spending and service levels. Evidence from 15 years of municipal boundary reform

14020-EEF: Kuper, G.H. and J.H. Veurink, Central bank independence and political pressure in the Greenspan era

14021-GEM: Samarina, A. and D. Bezemer, Do Capital Flows Change Domestic Credit Allocation?

14022-EEF: Soetevent, A.R. and L. Zhou, Loss Modification Incentives for Insurers Under ExpectedUtility and Loss Aversion
14023-EEF: Allers, M.A. and W. Vermeulen, Fiscal Equalization, Capitalization and the Flypaper Effect.

14024-GEM: Hoorn, A.A.J. van, Trust, Workplace Organization, and Comparative Economic Development.

14025-GEM: Bezemer, D., and L. Zhang, From Boom to Bust in de Credit Cycle: The Role of Mortgage Credit.

14026-GEM: Zhang, L., and D. Bezemer, How the Credit Cycle Affects Growth: The Role of Bank Balance Sheets.

14027-EEF: Bružikas, T., and A.R. Soetevent, Detailed Data and Changes in Market Structure: The Move to Unmanned Gasoline Service Stations.

14028-EEF: Bouwmeester, M.C., and B. Scholtens, Cross-border Spillovers from European Gas Infrastructure Investments.

14029-EEF: Lestano, and G.H. Kuper, Correlation Dynamics in East Asian Financial Markets.

14030-GEM: Bezemer, D.J., and M. Grydaki, Nonfinancial Sectors Debt and the U.S. Great Moderation.

14031-EEF: Hermes, N., and R. Lensink, Financial Liberalization and Capital Flight: Evidence from the African Continent.

14032-OPERA: Blok, C. de, A. Seepma, I. Roukema, D.P. van Donk, B. Keulen, and R. Otte, Digitalisering in Strafrechtketens: Ervaringen in Denemarken, Engeland, Oostenrijk en Estland vanuit een Supply Chain Perspectief.

14033-OPERA: Olde Keizer, M.C.A., and R.H. Teunter, Opportunistic condition-based maintenance and aperiodic inspections for a two-unit series system.

14034-EEF: Kuper, G.H., G. Sierksma, and F.C.R. Spieksma, Using Tennis Rankings to Predict Performance in Upcoming Tournaments

15001-EEF: Bao, T., X. Tian, X. Yu, Dictator Game with Indivisibility of Money

15002-GEM: Chen, Q., E. Dietzenbacher, and B. Los, The Effects of Ageing and Urbanization on China's Future Population and Labor Force

15003-EEF: Allers, M., B. van Ommeren, and B. Geertsema, Does intermunicipal cooperation create inefficiency? A comparison of interest rates paid by intermunicipal organizations, amalgamated municipalities and not recently amalgamated municipalities

15004-EEF: Dijkstra, P.T., M.A. Haan, and M. Mulder, Design of Yardstick Competition and Consumer Prices: Experimental Evidence

15005-EEF: Dijkstra, P.T., Price Leadership and Unequal Market Sharing: Collusion in Experimental Markets

15006-EEF: Anufriev, M., T. Bao, A. Sutin, and J. Tuinstra, Fee Structure, Return Chasing and Mutual Fund Choice: An Experiment

15007-EEF: Lamers, M., Depositor Discipline and Bank Failures in Local Markets During the Financial Crisis

15008-EEF: Oosterhaven, J., On de Doubtful Usability of the Inoperability IO Model

15009-GEM: Zhang, L. and D. Bezemer, A Global House of Debt Effect? Mortgages and Post-Crisis Recessions in Fifty Economies

15010-I&O: Hooghiemstra, R., N. Hermes, L. Oxelheim, and T. Randøy, The Impact of Board Internationalization on Earnings Management

15011-EEF: Haan, M.A., and W.H. Siekman, Winning Back the Unfaithful while Exploiting the Loyal: Retention Offers and Heterogeneous Switching Costs

15012-EEF: Haan, M.A., J.L. Moraga-González, and V. Petrikaite, Price and Match-Value Advertising with Directed Consumer Search

15013-EEF: Wiese, R., and S. Eriksen, Do Healthcare Financing Privatisations Curb Total Healthcare Expenditures? Evidence from OECD Countries

15014-EEF: Siekman, W.H., Directed Consumer Search

15015-GEM: Hoorn, A.A.J. van, Organizational Culture in the Financial Sector: Evidence from a Cross-Industry Analysis of Employee Personal Values and Career Success

15016-EEF: Te Bao, and C. Hommes, When Speculators Meet Constructors: Positive and Negative Feedback in Experimental Housing Markets

15017-EEF: Te Bao, and Xiaohua Yu, Memory and Discounting: Theory and Evidence

15018-EEF: Suari-Andreu, E., The Effect of House Price Changes on Household Saving Behaviour: A Theoretical and Empirical Study of the Dutch Case

15019-EEF: Bijlsma, M., J. Boone, and G. Zwart, Community Rating in Health Insurance: Trade-off between Coverage and Selection

15020-EEF: Mulder, M., and B. Scholtens, A Plant-level Analysis of the Spill-over Effects of the German *Energiewende*

15021-GEM: Samarina, A., L. Zhang, and D. Bezemer, Mortgages and Credit Cycle Divergence in Eurozone Economies

16001-GEM: Hoorn, A. van, How Are Migrant Employees Manages? An Integrated Analysis

16002-EEF: Soetevent, A.R., Te Bao, A.L. Schippers, A Commercial Gift for Charity

16003-GEM: Bouwmeerster, M.C., and J. Oosterhaven, Economic Impacts of Natural Gas Flow Disruptions

16004-MARK: Holtrop, N., J.E. Wieringa, M.J. Gijsenberg, and P. Stern, Competitive Reactions to Personal Selling: The Difference between Strategic and Tactical Actions

16005-EEF: Plantinga, A. and B. Scholtens, The Financial Impact of Divestment from Fossil Fuels

16006-GEM: Hoorn, A. van, Trust and Signals in Workplace Organization: Evidence from Job Autonomy Differentials between Immigrant Groups

16007-EEF: Willems, B. and G. Zwart, Regulatory Holidays and Optimal Network Expansion

16008-GEF: Hoorn, A. van, Reliability and Validity of the Happiness Approach to Measuring Preferences

16009-EEF: Hinloopen, J., and A.R. Soetevent, (Non-)Insurance Markets, Loss Size Manipulation and Competition: Experimental Evidence

16010-EEF: Bekker, P.A., A Generalized Dynamic Arbitrage Free Yield Model

16011-EEF: Mierau, J.A., and M. Mink, A Descriptive Model of Banking and Aggregate Demand

16012-EEF: Mulder, M. and B. Willems, Competition in Retail Electricity Markets: An Assessment of Ten Year Dutch Experience

16013-GEM: Rozite, K., D.J. Bezemer, and J.P.A.M. Jacobs, Towards a Financial Cycle for the US, 1873-2014

16014-EEF: Neuteleers, S., M. Mulder, and F. Hindriks, Assessing Fairness of Dynamic Grid Tariffs

16015-EEF: Soetevent, A.R., and T. Bružikas, Risk and Loss Aversion, Price Uncertainty and the Implications for Consumer Search

16016-HRM&OB: Meer, P.H. van der, and R. Wielers, Happiness, Unemployment and Self-esteem

16017-EEF: Mulder, M., and M. Pangan, Influence of Environmental Policy and Market Forces on Coal-fired Power Plants: Evidence on the Dutch Market over 2006-2014

16018-EEF: Zeng,Y., and M. Mulder, Exploring Interaction Effects of Climate Policies: A Model Analysis of the Power Market

16019-EEF: Ma, Yiqun, Demand Response Potential of Electricity End-users Facing Real Time Pricing

16020-GEM: Bezemer, D., and A. Samarina, Debt Shift, Financial Development and Income Inequality in Europe

16021-EEF: Elkhuizen, L, N. Hermes, and J. Jacobs, Financial Development, Financial Liberalization and Social Capital

16022-GEM: Gerritse, M., Does Trade Cause Institutional Change? Evidence from Countries South of the Suez Canal

16023-EEF: Rook, M., and M. Mulder, Implicit Premiums in Renewable-Energy Support Schemes

17001-EEF: Trinks, A., B. Scholtens, M. Mulder, and L. Dam, Divesting Fossil Fuels: The Implications for Investment Portfolios

17002-EEF: Angelini, V., and J.O. Mierau, Late-life Health Effects of Teenage Motherhood

17003-EEF: Jong-A-Pin, R., M. Laméris, and H. Garretsen, Political Preferences of (Un)happy Voters: Evidence Based on New Ideological Measures

17004-EEF: Jiang, X., N. Hermes, and A. Meesters, Financial Liberalization, the Institutional Environment and Bank Efficiency

17005-EEF: Kwaak, C. van der, Financial Fragility and Unconventional Central Bank Lending Operations

17006-EEF: Postelnicu, L. and N. Hermes, The Economic Value of Social Capital

17007-EEF: Ommeren, B.J.F. van, M.A. Allers, and M.H. Vellekoop, Choosing the Optimal Moment to Arrange a Loan

17008-EEF: Bekker, P.A., and K.E. Bouwman, A Unified Approach to Dynamic Mean-Variance Analysis in Discrete and Continuous Time

17009-EEF: Bekker, P.A., Interpretable Parsimonious Arbitrage-free Modeling of the Yield Curve

17010-GEM: Schasfoort, J., A. Godin, D. Bezemer, A. Caiani, and S. Kinsella, Monetary Policy Transmission in a Macroeconomic Agent-Based Model

17011-I&O: Bogt, H. ter, Accountability, Transparency and Control of Outsourced Public Sector Activities

17012-GEM: Bezemer, D., A. Samarina, and L. Zhang, The Shift in Bank Credit Allocation: New Data and New Findings

17013-EEF: Boer, W.I.J. de, R.H. Koning, and J.O. Mierau, Ex-ante and Ex-post Willingness-to-pay for Hosting a Major Cycling Event

17014-OPERA: Laan, N. van der, W. Romeijnders, and M.H. van der Vlerk, Higher-order Total Variation Bounds for Expectations of Periodic Functions and Simple Integer Recourse Approximations

17015-GEM: Oosterhaven, J., Key Sector Analysis: A Note on the Other Side of the Coin

17016-EEF: Romensen, G.J., A.R. Soetevent: Tailored Feedback and Worker Green Behavior: Field Evidence from Bus Drivers

17017-EEF: Trinks, A., G. Ibikunle, M. Mulder, and B. Scholtens, Greenhouse Gas Emissions Intensity and the Cost of Capital

17018-GEM: Qian, X. and A. Steiner, The Reinforcement Effect of International Reserves for Financial Stability

17019-GEM/EEF: Klasing, M.J. and P. Milionis, The International Epidemiological Transition and the Education Gender Gap

2018001-EEF: Keller, J.T., G.H. Kuper, and M. Mulder, Mergers of Gas Markets Areas and Competition amongst Transmission System Operators: Evidence on Booking Behaviour in the German Markets

2018002-EEF: Soetevent, A.R. and S. Adikyan, The Impact of Short-Term Goals on Long-Term Objectives: Evidence from Running Data

2018003-MARK: Gijsenberg, M.J. and P.C. Verhoef, Moving Forward: The Role of Marketing in Fostering Public Transport Usage

2018004-MARK: Gijsenberg, M.J. and V.R. Nijs, Advertising Timing: In-Phase or Out-of-Phase with Competitors?

2018005-EEF: Hulshof, D., C. Jepma, and M. Mulder, Performance of Markets for European Renewable Energy Certificates

2018006-EEF: Fosgaard, T.R., and A.R. Soetevent, Promises Undone: How Committed Pledges Impact Donations to Charity

2018007-EEF: Durán, N. and J.P. Elhorst, A Spatio-temporal-similarity and Common Factor Approach of Individual Housing Prices: The Impact of Many Small Earthquakes in the North of Netherlands

2018008-EEF: Hermes, N., and M. Hudon, Determinants of the Performance of Microfinance Institutions: A Systematic Review

2018009-EEF: Katz, M., and C. van der Kwaak, The Macroeconomic Effectiveness of Bank Bail-ins

2018010-OPERA: Prak, D., R.H. Teunter, M.Z. Babai, A.A. Syntetos, and J.E. Boylan, Forecasting and Inventory Control with Compound Poisson Demand Using Periodic Demand Data

2018011-EEF: Brock, B. de, Converting a Non-trivial Use Case into an SSD: An Exercise

2018012-EEF: Harvey, L.A., J.O. Mierau, and J. Rockey, Inequality in an Equal Society

2018013-OPERA: Romeijnders, W., and N. van der Laan, Inexact cutting planes for two-stage mixed-integer stochastic programs

2018014-EEF: Green, C.P., and S. Homroy, Bringing Connections Onboard: The Value of Political Influence

2018015-OPERA: Laan, N. van der, and W. Romeijnders, Generalized aplha-approximations for two-stage mixed-integer recourse models

2018016-GEM: Rozite, K., Financial and Real Integration between Mexico and the United States

2019001-EEF: Lugalla, I.M., J. Jacobs, and W. Westerman, Drivers of Women Entrepreneurs in Tourism in Tanzania: Capital, Goal Setting and Business Growth

2019002-EEF: Brock, E.O. de, On Incremental and Agile Development of (Information) Systems

www.rug.nl/feb