Conceptual modeling in agile information systems development

Antoni Olivé
Universitat Politècnica Catalunya-BarcelonaTech
Questions without definitive answers

Is conceptual modeling a necessary or an optional activity in information systems development?

What is it?

agile
Motivation

“The main purpose of conceptual modeling is to improve communication between the parties involved in the development process”

“Conceptual data modeling is an indispensable part of information system design and development”
Motivation

“...*might be* used to facilitate the design and implementation of an information system”

“Once you have the conceptual design, all the other design and implementation activities can and *should* be grounded in it...”
Manifesto for agile software development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

... Working software over comprehensive documentation ...

That is, while there is value in the items on the right, we value the items on the left more.
Dear professor,

I've started to develop my final career project in the company. Here they use agile development, and they do not generate any kind of documentation, and the requirements engineering and overall design are not very deep (I understand that this is their development method).

My idea of the project is to approach it in this way...
Objective of the talk

Is conceptual modeling a necessary or an optional activity in information systems development?
Implications

In (agile) information systems development

Practice
Do we need to perform conceptual modeling?

Teaching
Do we have to teach conceptual modeling?

Research
Which is the nature and role of conceptual modeling?
Outline

• Back to basics: the need of requirements
• What are the conceptual schemas? Do we really need them?
• The main driving force of formal conceptual schemas
• Conceptual modeling in agile development. Is it needed?
• Conclusions
You cannot develop a system unless you know its requirements.
The principle of necessity of requirements

To develop an information system it is necessary to define its requirements

Before software can be designed, programmed, coded, its requirements must first be reasonably well understood.

Are there exceptions?
Infinite monkey theorem

A monkey
• hitting keys at random on a typewriter keyboard
• for an infinite amount of time
• will almost surely type a given text,
• such as the complete works of William Shakespeare.
Infinite programmer theorem

A programmer
• hitting keys at random on a typewriter keyboard
• for an infinite amount of time
• will almost surely type a a program,
• that satisfies the stakeholders’ needs
Two types of requirements

- Functional
- Quality

The focus of this talk
The functions of an information system

- **Domain**
  - change
  - represents
  - changes

- **State**
  - Information system
  - action
  - output
Functional requirements

- Domain concepts represented in the IS
- Definitions
- Integrity Constraints
- Events/Actions:
  - Triggering conditions
  - Constraints
  - Effect on the state
  - Output
The principle of necessity of functional requirements

To develop an information system it is necessary to define its functional requirements
Outline

• Back to basics: the need of requirements

What are the **conceptual schemas**? Do we really need them?

• The main driving force of **formal** conceptual schemas

• Conceptual modelling in **agile** development. Is it needed?

• Conclusions
## Classification of functional requirements

<table>
<thead>
<tr>
<th></th>
<th>Implicit (Individual)</th>
<th>Explicit (Shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informal</td>
<td>Non-verifiable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-persistent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Persistent</td>
</tr>
<tr>
<td>Verifiable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-persistent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Persistent</td>
</tr>
<tr>
<td>Formal</td>
<td>(Verifiable)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-persistent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Persistent</td>
</tr>
</tbody>
</table>
Four key classifications of functional requirements

- Formality
- Verifiability
- Explicitness
- Persistency
## Formality

<table>
<thead>
<tr>
<th>Natural language</th>
<th>User stories</th>
<th>Structured natural language</th>
<th>UML/OCL</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informal</td>
<td>Semi-formal</td>
<td>Formal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Degree of formality of the language
Verifiability

Verifiable:
A person or a machine can check that the software meets the requirement

<table>
<thead>
<tr>
<th>Ambiguous requirement</th>
<th>Partly ambiguous requirement</th>
<th>OCL constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-verifiable</td>
<td>Verifiable</td>
<td></td>
</tr>
</tbody>
</table>
Explicitness

*Explicit*: Made public in some form

<table>
<thead>
<tr>
<th>Assumed requirements</th>
<th>SRS document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit</td>
<td>Explicit</td>
</tr>
</tbody>
</table>
Persistency

<table>
<thead>
<tr>
<th>Verbal</th>
<th>Non-persistent</th>
<th>Persistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper document</td>
<td>Computer file</td>
<td></td>
</tr>
</tbody>
</table>
## Classification of functional requirements

<table>
<thead>
<tr>
<th></th>
<th>Implicit (Individual)</th>
<th>Explicit (Shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Non-persistent</td>
<td>Persistent</td>
</tr>
<tr>
<td><strong>Informal</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-verifiable</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Verifiable</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Formal</strong></td>
<td>(Verifiable)</td>
<td></td>
</tr>
</tbody>
</table>

*Note: The shaded area indicates the classification of formal, verifiable requirements.*
Functional requirements to be implemented must be made verifiable

<table>
<thead>
<tr>
<th></th>
<th>Implicit (Individual)</th>
<th>Explicit (Shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-verifiable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verifiable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formal (Verifiable)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Functional requirements to be implemented must be made explicit**

<table>
<thead>
<tr>
<th></th>
<th>Implicit (Individual)</th>
<th>Explicit (Shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-persistent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistent</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Informal**: Non-verifiable
- **Verifiable**:
  - Non-persistent
  - Persistent

- **Formal** (Verifiable)
Conceptual schema = Explicit, verifiable functional reqs.

<table>
<thead>
<tr>
<th></th>
<th>Implicit (Individual)</th>
<th>Explicit (Shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-persistent</td>
<td></td>
<td>Persistent</td>
</tr>
<tr>
<td>Informal</td>
<td>Non-verifiable</td>
<td>Verifiable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formal</td>
<td>(Verifiable)</td>
<td></td>
</tr>
</tbody>
</table>
The principle of necessity of conceptual schemas

To develop an information system it is necessary to define its functional requirements.

To develop an information system it is necessary to define its conceptual schema.
“Official” definitions

...conceptual modelers:

(1) describe structure models in terms of entities, relationships, and constraints;

(2) describe behavior or functional models in terms of states, transitions among states, and actions performed in states and transitions; and

(3) describe interactions and user interfaces in terms of messages sent and received and information exchanged.

http://www.conceptualmodeling.org/ConceptualModeling.html
“Official” definitions

... conceptual-model diagrams:

• ...

• in some cases automatically generate (parts of) the software application.

http://www.conceptualmodeling.org/ConceptualModeling.html
Outline

• Back to basics: the need of requirements
• What are the conceptual schemas? Do we really need them?
  The main driving force of formal conceptual schemas
  • Conceptual modeling in agile development. Is it needed?
  • Conclusions
**Conceptual schemas: Formal or informal?**

<table>
<thead>
<tr>
<th></th>
<th>Implicit (Individual)</th>
<th>Explicit (Shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Informal</strong></td>
<td>Non-verifiable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-persistent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Persistent</td>
</tr>
<tr>
<td><strong>Verifiable</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Formal</strong> (Verifiable)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The diagram shows the relationship between conceptual schemas and their characteristics, with formal (verifiable) schemas being the most informative and persistent.
Formalization?
A cost/benefit analysis

Cost  Benefit
From requirements to implementation

User interface

Domain and database layer

Execution platform

- Manually obtained
- Automatically obtained

Design and construction

Model transformation

Verifiable

Formal
Outline

• Back to basics: the need of **requirements**

• What are the **conceptual schemas**? Do we really need them?

• The main driving force of **formal** conceptual schemas

Conceptual modeling in **agile** development. Is it needed?

• Conclusions
Manifesto for agile software development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.
Conceptual modeling in agile software development
Two views

Working software over comprehensive documentation

The focus of this talk
Functional requirements in agile development: User stories

In consultation with the customer or product owner, the team divides up the work to be done into functional increments called “user stories”. 
User stories

As a role I want feature so that reason

As a bank account holder
I want to be informed if my monthly balance is projected to go to zero or below
so that I can arrange for an overdraft
One thing the customer wants the system to do.

... Should be verifiable.
## User stories

<table>
<thead>
<tr>
<th>Informal</th>
<th>Non-verifiable</th>
<th>Verifiable</th>
<th>Explicit (Shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit</td>
<td>(Individual)</td>
<td></td>
<td>Non-persistent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Persistent</td>
</tr>
<tr>
<td>Formal</td>
<td>(Verifiable)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conceptual schema
The principle of necessity of conceptual schemas

in an agile way

To develop an information system it is necessary to define its conceptual schema
Alistair Cockburn’s Cooperative game principle

Software development is a cooperative game...
The **primary** goal of the game is to deliver useful, working software.
The **secondary** goal is to set up for the next game.
The next game may be:
- to alter or replace the system or
- to create a neighboring system.
### Conceptual schemas: Persistent or non-persistent?

<table>
<thead>
<tr>
<th></th>
<th>Implicit (Individual)</th>
<th>Explicit (Shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non-persistent</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Informal</td>
<td>Non-verifiable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Verifiable</td>
<td></td>
</tr>
<tr>
<td>Formal (Verifiable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Persistent</td>
</tr>
</tbody>
</table>

- Non-verifiable
- Verifiable
Persistent or non-persistent?
A life-cycle cost analysis

\[ CP = \text{Cost of initial documentation} + N \times \text{Average cost of updating documentation} \]

\( N \) : Number of system’s functional updates
Persistent or non-persistent?
A life-cycle cost analysis

\[ CP = \text{Cost of initial documentation} + N \times \text{Average cost of updating documentation} \]

\[ CNP = M \times \text{Average cost of recovering requirements} \]

N : Number of system’s functional updates
M : Number of times functional requirements are needed and not available
Persistent or non-persistent? A life-cycle cost analysis
Technical debt

... A metaphor referring to the eventual consequences of poor software development. The debt can be thought of as work that needs to be done before a particular job can be considered complete. If the debt is not repaid, then it will keep on accumulating interest, making it hard to implement changes later on.

Common causes of technical debt:

**Lack of documentation**, where code is created without necessary supporting documentation. That work to create the supporting documentation represents a debt that must be paid.
Non-persistent may be better when...

$$\text{CNP} = M \times \text{Average cost of recovering requirements}$$

- Short life cycle
- Few people
- Stable team
- Team remembers everything always

- Low cost

M : Number of times functional requirements are needed and not available
How to decrease the initial cost

May remain non-persistent requirements that:

• Can be easily observed in the system
• Can be easily recovered
• Are common-sense
• Are obvious (dependent)
Sometimes requirements are easily observed in the system

But not always
Agile conceptual schemas?

No results found for "agile conceptual schema".

Results for agile conceptual schema (without quotes):
Agile persistent conceptual schemas?

- Domain concepts represented in the IS
- Definitions
- Integrity Constraints
- Events/Actions:
  - Triggering conditions
  - Constraints
  - Effect on the state
  - Output

(*): When may not be known/accessible
(**): When not externally visible
(***): When not externally visible and independent
Outline

• Back to basics: the need of requirements
• What are the conceptual schemas? Do we really need them?
• The main driving force of formal conceptual schemas
• Conceptual modeling in agile development. Is it needed?

Conclusions
Conclusions (I)

The principle of necessity applies also to agile methods:

To develop an information system it is necessary to define its conceptual schema

Conceptual modeling is a necessary activity in information systems development
Conclusions (II)

Key distinctions in conceptual schemas:

- Informal/formal
  - Formalization enables automatic processing

- Non-persistent/persistent
  - Has non-obvious cost implications
  - Agile conceptual schemas may be an appropriate persistence degree.
Thanks for your attention