Domain Modeling
- Challenges in Requirements Engineering and System Design:
  Modeling versus Experiment-Based Approach
  with a Touch of Philosophy of Science

Manfred Broy
Philosophy of Science

Fundamental methods of science in capturing the real world

• Observation
  ◦ Experience based
  ◦ Experiment based
  Question: how to document and reason about observations

• Theory and modeling
  ◦ hypotheses
  ◦ deduction – reasoning
  ◦ verification

• Synthesis
  ◦ validation of theories
  ◦ making theories reality
The Theory and Scientific Basis of Software Engineering

• Software Engineering as a science of design and creation
  ◊ Engineering science

• Software engineering as an ultimate discipline of turning description into action/reality
  ◊ a description is not only a representation of knowledge about reality, it is at the same time part of reality
  ◊ blurring the classical distinction between knowledge and reality - knowledge as explicit part of reality

• Thus software engineering aims both at
  ◊ Understanding the world
  ◊ Finding solutions to problems
  ◊ Make the solutions operational

• In the end, software engineering is a weird mixture of a science of
  ◊ insight, perception, knowledge, cognition and
  ◊ design, creation, solution
  where the border between those is less strict than in any other discipline
The power of generalizing ideas, of drawing comprehensive conclusions from individual observations, is the only acquirement, for an immortal being, that really deserves the name of knowledge.

“Mary Wollstonecraft (1759–1797), British feminist. A Vindication of the Rights of Woman, ch. 4 (1792)
Domain

• The notion of a “domain” addresses
  ◊ a particular field of knowledge
  ◊ an area/field of application

• Domain knowledge includes
  ◊ terminology – propaedeutics / etymology
  ◊ rules, principles
  ◊ standards
  ◊ theory
  ◊ processes
  ◊ business
  ◊ experience
  ◊ state of the art, science, practice
  ◊ …
An industrial press system

Here's the machine

Press Controller Architecture
- Software
- Software

Here's the problem world

Actuators

Sensors

Press Mechanism & Doors

Operator Controls

Lamp & Switch

Operator

Here's the user interface

A slide due to Michael Jackson
Software systems and their domain of application

- A software systems is embedded into its domain of application or several domains of application.
- Requirements for a software systems cannot be captured domain of application.
- There is a multitude of aspects in a domain:
  - terminology and key notions and concepts
  - operational context information: systems and users in the environment and their behavior in terms of interaction with the SuC including business processes supported by the SuC
  - application domain concepts and rules
  - business rules
Relating Domain Specific Levels of Abstraction: Safety

We have to show

\[ \text{SRAL} \land \text{TATL} \Rightarrow \text{SRAL} \]

<table>
<thead>
<tr>
<th>SRAL: Safety Requirements: Abstract Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>aircraft_on_ground \land aircraft_moving \iff thrust_reversal_can_be_activated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SRTL: Safety Requirements: Technical Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>torque_on_wheels \iff thrust_reversal_can_be_activated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TATL: Translation: Abstract/technical Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>torque_on_wheels \iff aircraft_on_ground \land aircraft_moving</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Formalizing Domain Properties

• Domain properties are basis for software & systems requirements and their implementation
• This requires a formalization of domain properties
• To formalize properties of a domain item,
  ◇ A data model for the item has be created
    **Example:** Plane
  ◇ A logical predicate
    **Example:** reliable: Plane \(\rightarrow\) IB
• Note:
  ◇ A logical predicate can only be defined over a **mathematical set**
    **Example:** Define the set Plane of airplanes
  ◇ To form a set we have to formalize notions of the application domain
Process

Operational Context (OC)

User Interface

Physical and technical context

System under Consideration (SuC)

Context observations (CO)
A **system** has

- a system **boundary** that determines
  - what is part of the system and
  - what lies outside (called its **context**)

- an **interface** (determined by the system boundary), which determines,
  - what ways of interaction (actions) between the system and its context are possible (static or **syntactic interface**)
  - which behavior the system shows from view of the context (**interface behavior**, dynamic interface, interaction view)

- a structure and distribution addressing internal structure, given
  - by its structuring in sub-systems (**sub-system architecture**)
  - by its states and state transitions (**state view**, state machines)

- **quality** profile

- the views use a **data model**

- the views may be documented by adequate models
System and its context

- Physical World
- Context System
- System
- HMI
- Operational Context
- Cyberspace Services & Data
Sets of typed channels

\[ I = \{x_1 : T_1, x_2 : T_2, \ldots \} \]
\[ O = \{y_1 : T'_1, y_2 : T'_2, \ldots \} \]

Syntactic interface

\[(I \triangleright O)\]

data stream of type T

\[ \text{STREAM}[T] = \{\text{IN} \rightarrow T^*\} \]

Valuation of channel set Z

\[ \text{IH}[Z] = \{Z \rightarrow \text{STREAM}[T]\} \]

Interface behaviour for syn. interface (I \triangleright O)

\[ [I \triangleright O] = \{\text{IH}[I] \rightarrow \emptyset (\text{IH}[O])\} \]

Interface specification

\[ p : I \cup O \rightarrow IB \]

Represented as interface assertion S
Logical formula with channel names as variables for streams
Example: System interface specification

A broadcast component BC

<table>
<thead>
<tr>
<th>in</th>
<th>x: Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y: Data</td>
</tr>
<tr>
<td></td>
<td>x &gt;&gt; y</td>
</tr>
</tbody>
</table>

\[ x >> y = \]

\[ \forall d \in \text{Data}: d \text{ in } x \Leftrightarrow d \text{ in } y \]

\[ \land \]

\[ \forall d \in \text{Data}, t \in \text{IN}: d \text{ in } x.t \]

\[ \Leftrightarrow \]

\[ \forall t' \in \text{IN}: t' > t \Rightarrow \exists t'' \in \text{IN}: t'' \geq t' \land d \text{ in } y.t'' \]
Interface specification composition rule

F1

| in  | x1, z21: T |
| out | y1, z12: T |
| P1  |            |

F2

| in  | x2, z12: T |
| out | y2, z21: T |
| P2  |            |

F1 ⊗ F2

| in  | x1, x2: T |
| out | y1, y2: T |
| ∃ z12, z21: P1 ∧ P2 |
A safety view onto a system and its context

No Hazard:

\[ \text{OC} \land \text{SuC} \Rightarrow \text{No\_Incident}(\text{CO}) \]
Operational Context

OC

SoC

System under Consideration

Moving scopes: Context view

Context observations

System Spec:

\[ \text{SuC} \equiv (\text{OC} \Rightarrow \text{CO}) \]
Moving scopes:
System interface view

SoC
System under Consideration
Moving scopes: Architectural view

System under Consideration

Architecture Spec:
\[ A \land B \land C \land D \land E \land F \]

Architecture Correctness
\[ A \land B \land C \land D \land E \land F \Rightarrow \text{SuC} \]
Moving scopes: Sub-system view

Sub-system Spec:
\[ A \equiv (B \land C \land D \land E \land F \implies \text{SuC}) \]
Moving scopes: Sub-system view
Process

Operational Context

User Interface

Physical and technical context

System under Consideration

Context observations

System level glass box view
Example: Changing Scopes

Interface behaviour of the “outer” system:
\[ \text{CPS} = \text{S/A} \otimes \text{LSK} \otimes \text{PRC} \otimes \text{MMI} \]

Interface behaviour of the inner system:
\[ \text{LSK} \]

The architecture of the system:
\[ \text{SYS} = \text{PHYC} \otimes [\times] \text{USER} \otimes [\times] \text{SURSYS} \otimes [\times] \text{CPS} \]

Three different system scopes and two different contexts
- The empty context for SYS
- The physical context, the user, surrounding system for the CPS
- The (sensor/actuator \( \otimes \) physical context), the (MMI \( \otimes \) User) and the (protocol \( \otimes \) surrounding-system) for the logical system kernel
Summary and Outlook

• Rigorous framework of modelling
  ◊ to capture logical and probabilistic properties at different levels of abstraction
  ◊ modelling that can be used both for system specification, design and implementation, for verification including test case generation, for safety analysis as well as for diagnoses.
• model the system as well as its operational context
• distinguish and to model intrinsic as well as extrinsic hazards
• validating the specification carefully to make sure that hazards are not implied by it – system safe as specified
• apply all kinds of automatic analysis and verification techniques to deal with functional safety
  ◊ quality of functional safety analysis depends on the expressive power and the adequate application of the modelling techniques and methods.
To what degree have we achieved predictability, scalability based on regularities, systematic reuse, and other typical engineering characteristics?

How should we use “empirical evidence” to get a handle on predictability via “process-product” relationships?

How should software engineering be positioned as academic discipline (part of computer science, or separate like other engineering disciplines)?

What are current and future “big challenges” in our field?