Seamless Model- and Method-Based Software & Systems Engineering

Scientific Foundations

Manfred Broy

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany
Challenges in Software & Systems Development

- Cost
- Complexity
  - Size
  - Complex relationships
- Systematic development
  - Process
  - System quality
- Key Challenges
  - Requirements
    - Functional
    - Quality
    - Constraints
  - Specification
    - Functionality
  - Architecture
    - Interfaces
    - Components
  - Quality
- Terminology
- Modelling concepts
  - artefacts
  - description techniques
  - semantic foundations
- Development methods
- Artefact models
  - relationship between artefacts
- Tools
  - Automation
Perspectives

• Front loading
  ◊ Emphasis on requirements, specification, and architecture
  ◊ Early quality control
• Domain engineering
  ◊ Concentration on domain and use specific requirements
  ◊ Use case
• Artefact orientation
  ◊ Document every development artefact in a repository
  ◊ Define relationships (tracing) and rules of consistency
• Software & system evolution
• Product line engineering
  ◊ Reuse
  ◊ Systematic generation of software
Approach: foundations by theory

Idea

• develop theories in terms of mathematics and logics that capture
• essential concepts in software & systems engineering
• reflects the terminology
• proves or disproves ideas and concepts
• validates/verifies methods
• can be the basis of automation
• supports views
  ◦ abstraction
  ◦ structuring
Systems: the model

Systems
• boundaries
• typed channels
  \[I = \{x_1 : T_1, x_2 : T_2, \ldots\}\]
  \[O = \{y_1 : T'_1, y_2 : T'_2, \ldots\}\]
• syntactic interface
  \[(I \gg O)\]
• streams
  \[\text{STREAM}[T] = \{\text{IN} \rightarrow T^*\}\]
• histories for channels set \(C\)
  \[\text{IH}[C] = \{C \rightarrow \text{STREAM}[T]\}\]
• interface behaviour
  \[[I \gg O] = \{\text{IH}[I] \rightarrow \wp(\text{IH}[O])\}\]
Three levels of specification

• System level requirements (functional requirements)
  ◊ a list of requirements in terms of system properties

• System level functional specification
  ◊ a decomposition of the system functionality into a hierarchy of (sub-)functions
  ◊ a specification of the (sub-)functions by properties
  ◊ feature interactions specified via a mode concept

• Architecture specification
  ◊ a decomposition of the system into a sub-systems (components)
  ◊ their connections via their sub-interfaces
  ◊ interface specification by interface properties
Three levels of specification: examples

• System level requirements (functional requirements)
  ◊ “the car must not accelerate its speed without users control”

• System level functional specification
  ◊ “the acc (adaptive cruise control) accelerates the car up to the speed selected by the user, provided no obstacles are recognized in front”

• Architecture specification
  ◊ “the radar signal based sensor measures the distance in m/s to the car in front and sends it to the acc monitor every 100 ms”
Three levels of specification: logical model

• System level requirements (functional requirements)
  \[ R = \land \{ R_i : 1 \leq i \leq n \} \]

• System level functional specification
  \[ Q = \land \{ Q_i : 1 \leq i \leq m \} \]

• Architecture specification
  \[ P = \land \{ P_i : 1 \leq i \leq k \} \]

• Correctness
  Functional specification:
  \[ Q \Rightarrow R \]

  Architecture (let be \( m_1, \ldots \) mode channels):
  \[ P \Rightarrow \exists m_1, \ldots : Q \]
System level requirements specification

Specification template

SysSpc

<table>
<thead>
<tr>
<th>in</th>
<th>$x_1: T_1$, ..., $x_m: T_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>$y_1: T'_1$, ..., $y_n: T'_n$</td>
</tr>
</tbody>
</table>

$\land \{ R_i : 1 \leq i \leq n \}$

The $R_i$ express functional requirements and are called interface assertions
Example: System level requirements specification

A transmission component **FairMIX**

<table>
<thead>
<tr>
<th>in</th>
<th>x, z: T</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y: T</td>
</tr>
</tbody>
</table>

\[ \forall m \in T: \{m\}#x + \{m\}#z = \{m\}#y \]

\[ \wedge \forall t \in \text{IN}, m \in T: \{m\}#x\downarrow t + \{m\}#z\downarrow t \geq \{m\}#y\downarrow t+n \]
System level requirements

• The system interface behaviour $F$ is specified by the system requirements specification

$$R = \{R_i : 1 \leq i \leq n\}$$

where the $R_i$ are interface assertions
Functional view: Functional decomposition

• The system interface behaviour $F$ as specified by the system requirements specification $R = \{R_i: 1 \leq i \leq n\}$ is structured
  ◊ into a set of sub-interfaces for sub-functions $F_1, \ldots, F_k$
  ◊ that are specified independently by introducing a number of mode channels to capture feature interactions
  ◊ each $F_i$ sub-function is described by a syntactic interface and an interface assertion $Q_i$ such that

$$\wedge \{Q_i: 1 \leq i \leq k\} \Rightarrow R$$
Sub-types between interfaces

For syntactic interfaces \((I \to O)\) and \((I' \to O')\) where

\[ I' \text{ subtype } I \] and \[ O' \text{ subtype } O \]

we call \((I' \to O')\) a sub-type of \((I \to O)\) and write:

\[(I' \to O') \text{ subtype } (I \to O)\]
Combination of sub-functions

The combination of sub-functions

\[ F_1 \in \text{IF}[I_1 \triangleright O_1], \ F_2 \in \text{IF}[I_2 \triangleright O_2] \]

into a super-function

\[ F_1 \oplus F_2 \in \text{IF}[I_1 \cup I_2 \triangleright O_1 \cup O_2] \]

such that both \( F_1 \) and \( F_2 \) are sub-functions of \( F_1 \oplus F_2 \).
Combining Functions

Given two functions $F_1$ and $F_2$ in isolation

We want to combine them into a function $F_1 \oplus F_2$
Combining Functions

Their isolated combination

\[ F_1 \oplus F_2 \]

\[ F_1 \]

\[ F_2 \]

\[ I_1 \quad I_2 \]

\[ O_1 \quad O_2 \]
Combining Functions

If the two services $F_1$ and $F_2$ have feature interaction we typically get:

We explain the functional combination $F_1 \otimes F_2$ as a refinement step.
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
The steps of function combination

Given the isolated function $F_1$

We construct a refinement $F'_1$

And combine $F'_1$ with a refinement $F'_2$ of $F_2$
Applying projections - functional slicing

- identifying sub-functions - functional slicing
- identifying sub-interface behaviour
- Given some behaviour

\[ F \in [I\rightarrow O]\]

a set of behaviours

\[ F_k \in [I'_k\rightarrow O'_k]\]

with

\[ [I_k\rightarrow O_k] \text{ subtype } [I'_k\rightarrow O'_k]\]

\[ I = \bigcup \{ I_k : 1 \leq k \leq m \}, \quad O = \bigcup \{ O_k : 1 \leq k \leq m \}\]

is called a functional decomposition if

\[ F = \otimes \{ F_k : 1 \leq k \leq m \}\]
## Relation views: tracing

<table>
<thead>
<tr>
<th>Interface assertion</th>
<th>Safety</th>
<th>Priority</th>
<th>Component</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>Yes</td>
<td>high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R_2</td>
<td>No</td>
<td>medium</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>no</td>
<td>low</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram showing relation views and tracing](image)
Relating logical views

Let $p$ be a property and $R$ be a set of properties

- a subset $R' \subseteq R$ is called **guarantor** for $p$ in $R$ if
  \[ \wedge R' \Rightarrow p \]

**Classifying guarantors**

- A guarantor $R'$ for $p$ is called **minimal**, if every strict subset of $R'$ is not a guarantor.
- A minimal guarantor is called **unique** if there does not exist a different minimal guarantor.
- A property $q \in R$ is called **weak guarantor** for $p$ in $R$ if it occurs in some minimal guarantor of $p$ in $R$.
- A property $q \in R$ is called **strong guarantor** for $p$ in $R$ if it occurs in every guarantor of $p$ in $R$.

*Cf. Primimplikanten a la Quine*
**Relationship: req spec to function spec - tracing**

<table>
<thead>
<tr>
<th></th>
<th>system level reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R₁</td>
</tr>
<tr>
<td>sub-function reqs</td>
<td></td>
</tr>
<tr>
<td>Q₁</td>
<td></td>
</tr>
<tr>
<td>Q₂</td>
<td></td>
</tr>
<tr>
<td>Q₃</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Qₙ</td>
<td></td>
</tr>
</tbody>
</table>

**Red:** Qᵢ is strong guarantor of Rⱼ  
**Yellow:** Qᵢ is weak guarantor of Rⱼ  
**Green:** Qᵢ is not a weak guarantor of Rⱼ
Architecture

- Composition $F = F_1 \otimes F_2 \otimes F_3$
### Relationship: architecture to requirements

<table>
<thead>
<tr>
<th>sub-system reqs</th>
<th>R₁</th>
<th>R₂</th>
<th>R₃</th>
<th>R₄</th>
<th>R₅</th>
<th>R₆</th>
<th>R₇</th>
<th>R₈</th>
<th>R₉</th>
<th>...</th>
<th>...</th>
<th>...</th>
<th>Rₖ</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pₙ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Red**: Pᵢ is strong guarantor of Rⱼ  
**Yellow**: Pᵢ is weak guarantor of Rⱼ  
**Green**: Pᵢ is not a weak guarantor of Rⱼ
Relating components with system level functional specs

• Given a
  ◊ Functional structuring of the system level functionality
  ◊ A component architecture
  ◊ A functional decomposition of each of the components

• we relate
  ◊ each sub-function at the system level functionality with the component level
  ◊ each sub-function at the system level functionality with the sub-functions at the component level
**Relationship: architecture to functional spec**

<table>
<thead>
<tr>
<th>sub-system reqs</th>
<th>Sub-function reqs</th>
<th>Q₁</th>
<th>Q₂</th>
<th>Q₃</th>
<th>Q₄</th>
<th>Q₅</th>
<th>Q₆</th>
<th>Q₇</th>
<th>Q₈</th>
<th>Q₉</th>
<th>...</th>
<th>...</th>
<th>Qₖ</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pₙ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Red**: $P_i$ is strong guarantor of $Q_j$
- **Yellow**: $P_i$ is weak guarantor of $Q_j$
- **Green**: $P_i$ is not a weak guarantor of $Q_j
Basics: What we need

• Modelling theory for
  ◦ Systems
    • interface specifications
    • architectures
  ◦ Quality
• Comprehensive architecture
  ◦ Levels of abstraction
  ◦ Relationships between levels (tracing)
• Artefact model
  ◦ Structure of work products
  ◦ Tailoring
• Tool support
  ◦ Artefact based
  ◦ Automation
What we need ...
Conclusion

- Software engineering is much more than producing software by writing programs
  - Domain modelling
- Software cannot be better than its requirements - requirements engineering
  - must be based on domain knowledge
  - requires making implicit knowledge explicit
  - asks for modelling domain knowledge explicitly by modelling techniques
- Software contains domain knowledge often implicitly
  - modelling techniques can make it explicit
- Software validation & verification requires
  - to make domain knowledge explicit
  - to relate to software structuring