Modular Construction of Systems and Layered System Architectures by Rich Subsystem Specifications of Cyber-Physical Systems

Manfred Broy

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany
From mechatronics to embedded systems ...

Mechatronics (Wikipedia)
“Mechatronics is a multidisciplinary field of engineering, that is to say, it rejects splitting engineering into separate disciplines. Originally, mechatronics just included the combination of mechanics and electronics, hence the word is a combination of mechanics and electronics ...”

Embedded System (Wikipedia)
“An embedded system is a computer system designed for specific control functions within a larger system, often with real-time computing constraints.”
Cyber-Physical System (US NSF):
“Cyber-physical systems (CPS) are engineered systems that are built from and depend upon the synergy of computational and physical components. Emerging CPS will be coordinated, distributed, and connected, and must be robust and responsive.
The CPS of tomorrow will need to far exceed the systems of today in capability, adaptability, resiliency, safety, security, and usability.”
The acatech position on CPS

“Cyber-physical systems are systems with embedded software (as part of devices, buildings, means of transport, transport routes, production systems, medical processes, logistic processes, coordination processes and management processes), which:

- directly record physical data using sensors and affect physical processes using actuators;
- evaluate and save recorded data, and actively or reactively interact both with the physical and digital world;
- are connected with one another and in global networks via digital communication facilities (wireless and/or wired, local and/or global);
- use globally available data and services;
- have a series of dedicated, multimodal human-machine interfaces.”
Cyber-physical systems: key properties and challenges

- **Physicallity**
  - real time
  - probability
  - location awareness
  - context awareness
  - real world awareness
  - ...

- **Connectivity**
  - connected to cloud services

- **Systems of systems**
  - Sub-system (de)composition
  - Service (de)composition

- **Interoperability**
  - Service platforms

- **Openness**
  - security

- **HMI**
  - Human Centric Engineering

- **Dynamic systems**
  - Dynamic interfaces
  - Dynamic architectures
  - Dynamic change of behavior (adaptivity)

- **Mobile systems**
  - space awareness

- **Uncertainty**
Cyber-physical systems: innovation

The two sides of CPS

The internet becomes real world aware

Embedded systems get connected to cloud services
From closed embedded systems to open systems connected to the cloud

Traditional embedded systems
- closed
- real time
- connected to the physical
- reliable
- high safety reqs
- low security reqs

Services in the cloud
- open
  - open interfaces
- restricted availability
- easy extendibility
- high interoperability
- low safety reqs
- high security reqs

“Smart” systems - Cyber-physical systems: focus points of innovation
- adaptive
- context aware
- autonomous

- big data
- open interfaces
- dynamic
- tactile internet
What is seamless model based development?

- Development by a chain of models
  - High expressive power
  - Clear structure of role of models
  - All aspects in the development captured by models
  - Tight integration in the artifact models

- Development steps by well-defined relationship between modes
  - Refinement
  - Decomposition
  - Change of scope

- Extended tool support
  - High automation
  - All artifacts in tools and comprehensive data base (development back bone)
Essentials for Modeling Concepts for Seamless Modeling

- Expressive and tractable modeling concepts
- Clear understanding of relationship between modeling concepts
- Notion and understanding of integration of modeling concepts
- Tracing between the elements of modeling concepts
- Expressive power: relevant properties expressible
- Modularity and composition of modeling concepts
- Variety of forms of representation
  - Mathematical models
  - Logic
  - Intuitive and tractable representation: graphics, diagrams, tables, ...
Basic System Modeling Concepts
System and its context

- HMI
- Physical World
- Cyberspace Services & Data
- Context System
- Operational Context
Basic System Notion: What is a discrete system (model)

A system has

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

- an interface (determined by the system boundary), which determines,
  - what ways of interaction (actions) between the system und 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 und state transitions (state view, state machines)

- quality profile

- the views use a data model

- the views may be documented by adequate models
Discrete systems: the modeling theory – interface behaviour as key concept

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

\[(I \rightarrow O)\]

data stream of type \( T \)

\[ \text{STREAM}[T] = \{\text{IN}\backslash \{0\} \rightarrow T^*\} \]

valuation of channel set \( C \)

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

interface behaviour for syn. interface \((I \rightarrow O)\)

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

interface specification

\[ p: \text{IH}[I] \times \text{IH}[O] \rightarrow \text{IB} \]

represented as interface assertion \( S \)

logical formula with channel names as variables for streams

Modeling Interface Behavior
System interface behaviour - causality

(I \rightarrow O) \quad \textit{syntactic interface} \textit{with set of input channels } I \textit{ and of output channels } O

F : IH[I] \rightarrow \wp(IH[O]) \quad \textit{semantic interface} \text{ for } (I \rightarrow O) \text{ with } \textit{timing property addressing strong causality}

(let x, z \in IH[I], y \in IH[O], t \in \mathbb{N}):

\[ x \downarrow t = z \downarrow t \Rightarrow \{y \downarrow t+1: y \in F(x)\} = \{y \downarrow t+1: y \in F(z)\} \]

\[ x \downarrow t \quad \text{prefix of history } x \text{ of length } t \]

A system shows a \textit{total behavior}

Component interface
Example: System interface specification

A transmission component TMC

\[
\text{TMC}
\begin{array}{l}
\text{in} \ x: T \\
\text{out} \ y: T \\
\text{in}\ x \sim y
\end{array}
\]

\[x \sim y \equiv (\forall m \in T: m#x = m#y)\]

Specifying interface assertion
Verification: Proving properties about specified systems

From the interface assertions we can prove

• Safety properties

\[ m\#y > 0 \land y \in TMC(x) \Rightarrow m\#x > 0 \]

• Liveness properties

\[ m\#x > 0 \land y \in TMC(x) \Rightarrow m\#y > 0 \]
Verification: adding causality – consistent time flow

From the interface assertions we can derive properties!

**Specification:**

\[ y \in TMC(x) \Rightarrow (\forall m \in T: m#x = m#y) \]

**Strong causality:**

\[ x \downarrow t = z \downarrow t \Rightarrow \{y \downarrow t + 1: y \in TMC(x)\} = \{y \downarrow t + 1: y \in TMC(z)\} \]

From which by choosing \( z \) such that

\[ \forall m \in T: m#(z \uparrow t) = 0 \]

we conclude (note then \( x \downarrow t = z \downarrow t \Rightarrow m#(x \downarrow t) = m#z \)) and thus

\[ x \downarrow t = z \downarrow t \Rightarrow \]

\[ \{y \downarrow t + 1: y \in TMC(x)\} = \{y \downarrow t + 1: y \in TMC(z)\} = \{y \downarrow t + 1: y \sim z\} \]

and thus

\[ y \in TMC(x) \Rightarrow \forall t \in \text{Time}, m \in T: m#(y \downarrow t + 1) \leq m#(x \downarrow t) \]
Example: TMC with Timing Restrictions

\[
\begin{align*}
\forall t \in \mathbb{IN}: \forall m \in T: \\
m#(y\downarrow t + \text{delay}) & \leq m#(x\downarrow t) \leq m#(y\downarrow t + \text{delay} + \text{deadline})
\end{align*}
\]
A flexible model of time

• Time is a key issue in embedded systems:

• Dealing with timing properties
  ◊ Specification
  ◊ Analysis
  ◊ Verification
    • Analysis
    • Testing
    • Model checking
    • Deduction based verification

• Transforming time

• Dedicated models of time
  ◊ Micro/Macro Step
  ◊ Perfect synchrony
  ◊ Scheduling

• Abstractions

See:
M. Broy: Relating Time and Causality in Interactive Distributed Systems.
Continuous systems: the model

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

$(I \mid O)$

continuous data stream of continuous type $M$

$\text{ConSTREAM}[T] = \{IR_+ \rightarrow M\}$

valuation of channel set $C$

$CIH[C] = \{C \rightarrow \text{ConSTREAM}[T]\}$

interface behaviour for syn. interface $(I \mid O)$

$[I \mid O] = \{CIH[I] \rightarrow \varnothing(CIH[O])\}$

interface specification

$p: CIH[I] \times CIH[O] \rightarrow IB$

represented as interface assertion $S$

logical formula with channel names as variables for continuous streams

Example: TMC with Probability Restrictions

\[
\begin{align*}
\text{TMC} \\
\text{in} & \quad x : T \\
\text{out} & \quad y : T \\
\forall t & \in \mathbb{IN} : \forall m \in T : \\
P(m#(x \downarrow t) \leq m#(y \downarrow t + \text{delay} + \text{deadline})) & \geq 0.8
\end{align*}
\]
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 \( T \)

\[ \text{STREAM}[T] = \{ \text{IN}\backslash\{0\} \rightarrow T^* \} \]

valuation of channel set \( C \)

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

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

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

interface specification represented as interface assertion \( S \)

that talk about the probability of the correctness of interface assertions

Extensions of the model

• Physical Aspects & Properties: Rich Models
  ◊ Space
  ◊ Geometry
  ◊ Temperature
  ◊ ...

Structure: Composition and Decomposition
Modularity: Rules of compositions for interface specs

\[
\begin{align*}
F_1 & \otimes F_2 \\
\text{in} & \quad x_1, z_{12}: T \\
\text{out} & \quad y_1, z_{12}: T \\
S_1 & \\
F_2 & \\
\text{in} & \quad x_2, z_{12}: T \\
\text{out} & \quad y_2, z_{21}: T \\
S_2 & \\
\exists & \quad z_{12}, z_{21}: S_1 \wedge S_2
\end{align*}
\]
Forming Architectures

Architecture Behaviour

\[ SF_1 \otimes SF_2 \otimes SF_3 \]

Architecture Spec

\[ C_1 \land C_2 \land C_3 \]

Architecture Correctness

\[ C_1 \land C_2 \land C_3 \Rightarrow SysSpec \]
Probabilistic Behavior Composition

Probabilistic behavior

\[ F: \mathcal{P}(\mathbb{I}[I]) \rightarrow (\mathcal{P}(\mathbb{I}[O]) \rightarrow [0:1]) \]

We write for \( X \subseteq \mathbb{I}[I], \ Y \subseteq \mathbb{I}[O] \)

\[ F(X)[Y] \]

for the probability

that the output of \( F \) is in set \( Y \) provided the input is in set \( X \)
Probabilistic Behavior Composition

Given probabilistic behaviors

\[ F_1: \mathcal{P}(\mathcal{I}[I_1]) \rightarrow (\mathcal{P}(\mathcal{I}[O_1]) \rightarrow [0:1]) \]
\[ F_2: \mathcal{P}(\mathcal{I}[I_1]) \rightarrow (\mathcal{P}(\mathcal{I}[O_2]) \rightarrow [0:1]) \]

Where \( O_1 \cap O_2 = \emptyset \); we define

\[ I = I_1 \setminus O_2 \cup I_2 \setminus O_1, \quad O = (O_1 \cup O_2) \setminus Z, \]
\[ Z = (I_1 \cap O_2) \cup (I_2 \cap O_1) \] shared channels

Independent composition: define

\[ G: \mathcal{P}(\mathcal{I}[I_1 \cup I_2]) \rightarrow (\mathcal{P}(\mathcal{I}[O_1 \cup O_2]) \rightarrow [0:1]) \]

by

\[ G(X)[Y] = F_1(X|_{I_1})[\{y|_{O_1}: y \in Y\}] \times F_2(X|_{I_2})[\{y|_{O_2}: y \in Y\}] \]

Assuming probabilistic independence
Probabilistic Behavior Composition

Given $G$ we specify $F = F_1 \otimes F_2$

$$F: \emptyset (\mathbb{I}[I]) \rightarrow (\emptyset (\mathbb{I}[O]) \rightarrow [0:1])$$

by (here $y \in \mathbb{I}[O_1 \cup O_2]$)

$$F(X)[Y] = G(X')[\{y'|_O: \exists x' \in X': y'|_O \in Y \land x'|_Z = y'|_Z\}]$$

where $X' = \{x: x|_I \in X\}$.
Functional View: Functional Decomposition
Combining Functions

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

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

Their isolated combination

\[ F_1 \otimes F_2 \]

\[ F_1 \]

\[ F_2 \]

\[ I_1 \]

\[ I_2 \]

\[ O_1 \]

\[ O_2 \]
Combining Functions

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

We explain the functional combination $F_1 \otimes F_2$ as a refinement step.
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 $F_2$

Function Hierarchy

\[ F_{1, \ldots, n} \]

\[ \begin{align*}
B_1 & \quad B_2 & \quad \cdots & \quad B_n \\
F_{1,2} & \quad & \cdots & \quad \\
F_1 & \quad B_1 & \quad B_2 & \quad \cdots & \quad B_n & \quad F_{n-1,n} \\
F_2 & \quad & \cdots & \quad & \cdots & \quad \\
F_1 & \quad B_1 & \quad B_2 & \quad \cdots & \quad B_n & \quad F_{n-1,n} \\
F_k & \quad B_k & \quad B_{k+1} & \quad \cdots & \quad B_n & \quad F_n \\
F_{k+1} & \quad & \cdots & \quad & \cdots & \quad \\
F_k & \quad B_k & \quad B_{k+1} & \quad \cdots & \quad B_n & \quad F_n \\
\end{align*} \]

\( \text{subservice relation} \)

\( \text{channels of mode types} \)
Relational view: Tracing – The power of logics

<table>
<thead>
<tr>
<th>Function</th>
<th>Safety</th>
<th>Priority</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₁</td>
<td>Yes</td>
<td>high</td>
<td></td>
</tr>
<tr>
<td>A₂</td>
<td>No</td>
<td>medium</td>
<td></td>
</tr>
</tbody>
</table>

Modular Model Based System Development
System Specification

$S \xleftarrow{\text{architecture design}} C_1 \otimes C_2 \otimes C_3$

Validation

Informal requirements

System delivery

$R \xrightarrow{\text{system verification}} S$

Integration

$R = R_1 \otimes R_2 \otimes R_3$

Implementation

$R_1 \Rightarrow C_1 \quad R_2 \Rightarrow C_2 \quad R_3 \Rightarrow C_3$

Verification

Architecture

design

Manfred Broy
Extended Notions of Correctness: Reliability
Correctness and reliability ...

And what about requirements specification?

• Correctness does not make sense without specifications!
• Reliability needs also notions of correctness!

However for cyber-physical system specification and correctness is a bit more tricky ...

• Time
• Probability
• Precision
• Uncertainty of the physical world
• ...
The challenge: uncertainty and correctness of software/systems

- Classical: „sharp“ correctness – black or white
  - a system/program is correct or not
- **Unsharp correctness:**
  - Correct to a *certain degree*
  - Correct with a certain *probability*
  - Correct *over a certain time*
  - Correct in some *fuzzy way*

**Challenge**

- specification
- verification

in der presence von unsharpness/uncertainty
Formalizations of unsharp correctness

Given: set $T \subseteq \mathcal{IH}[I]$ of correct output histories

- **Classical correctness:**
  \[ \text{output } t' \in \mathcal{IH}[I] \text{ is correct, iff } t' \in T \]

- **Extension:** output $t'$ more correct than output $t''$
  Define distance $d(t, t')$ between output streams:
  \[ t' \text{ is more correct as } t'' \text{ iff } \min \{ d(t, t'): t \in T \} < \min \{ d(t, t''): t \in T \} \]

- **result $t' \in \mathcal{IH}[I]$ is correct with a certain probability:**
  \[ P[t' \in T] > 0.9 \]
  \[ P[\min \{ d(t, t'): t \in T \} < 0.1] > 0.9 \]

- **Fuzzy:** result $t' \in \mathcal{IH}[I]$ is roughly correct – formalized in fuzzy logic
Dependability

Comprehensive view **dependability**:

- **Availability** - readiness for sufficiently correct service
- **Reliability** - continuity of correct service
- **Safety** - absence of catastrophic consequences on the user(s) and the environment
- **Security** - Integrity - absence of improper system alteration/degree of resistance to or protection from vulnerability
- **Maintainability** - ability for a process to undergo modifications and repairs
Rich Interface Specifications

In a rich interface specification we speak about several views

Example: Add probability

Given:

• **logical interface behavior** for syn. interface \((I \rightarrow O)\)
  
  \[ \{IH[I] \rightarrow \emptyset (IH[O])\} \]

• **probabilistic** interface behavior for syn. interface \((I \rightarrow O)\)
  
  \[ \{IH[I] \rightarrow PD [ \emptyset (IH[O]) ] \} \]

• **interface specification** by an interface assertion \(q(x, y)\)

• specify for each input history \(x = a\) probability distributions
  \(P(y|a)\) on the set of output histories
  
  \[ \{y: q(a, y)\}\]
Rich Specifications

Specifying system behavior and functionality:

• Syntactic interface

• Logical system behavior
  ◊ Data flow
  ◊ Timing
  ◊ Discrete and/or continuous

• Probabilistic system behavior

• Quality aspects
  ◊ Quality of service: reliability – availability/correctness
  ◊ Performance
  ◊ Safety
  ◊ Security
  ◊ ...

• Physical behavioral aspects
Service Layers and Service Stacks
A service layer is a service with syntactic interface \((I \cup O' \rightarrow I' \cup O)\) structured into an provided ("exported") service \((I \rightarrow O)\) requested ("imported") service \((I' \rightarrow O')\).

We assume \(I \cap O' = \emptyset\) and \(O \cap I' = \emptyset\).

A service layer is a service with the interface behavior 
\[
L: \mathcal{H}(I \cup O') \rightarrow \mathcal{P}(\mathcal{H}(I' \cup O))
\] 
where both input and output actions are disjoint sets.
We denote the syntactic interface of a service layer by

\[(I \mapsto O/O' \mapsto I')\] syntactic service layer interface

The service layer is a service

\[L \in [I \cup O' \mapsto O \cup I']\]
Specifying Service Layers

To specify a service layer we specify two services:

The provided service \( F \in [I \rightarrow O] \)

The required service \( R \in [I' \rightarrow O'] \)

A layer provides the service \( F \) under the condition that it gets the service \( R \) from “below”.

Note that

\( R \) does not specify the service of the layer as provided by \( L \) but the required service.

Given \( F \) and \( R \) we denote the layer \( L \) that provides service \( F \) provided service \( R \) is offered as an auxiliary service by \( F//R \)
Specifying Service Layers

Layer

\[ L = F//R \]

is specified as follows (for \( x \in IH(I \cup O') \))

\[ L(x) = \{ y \in IH(I' \cup O): x|O' \in R(y|I') \Rightarrow y|O \in F(x|I) \} \]

This expresses that

- if the service provided from “below” is correct as required and specified by \( R \) than the offered service is as specified by \( F \).
- Note that this is written in the pattern of an assumption/commitment specification.
Composing Layers with Services

Given a service $R'$ as requested from below

\[ R' \in [I' \rightarrow O'] \quad \text{imported service} \]

and a service layer

\[ L \in [I \rightarrow O/O' \rightarrow I'] \]

where \[ L = F//R \]

with given \[ F \in [I \rightarrow O] \] and \[ R \in [I' \rightarrow O'] \]

We get the composition of layer $L$ with service $R'$

\[ L \otimes R' \in [I \rightarrow O] \text{composition of layer } L \text{ with service } R' \]
Service Refinement

Given services

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

\( F' \) is called a refinement of \( F \) iff

\[ \forall x \in \mathcal{H}(I): F'(x) \subseteq F(x) \]

then we write

\[ F \rightarrow F' \]
Composing Layers with Services

$L \otimes R' \in [I \rhd O]$ is a refinement of the provided service $F \in [I \rhd O]$

$F \Rightarrow L \otimes R'$

provided service $R'$ is a refinement of the requested service $R$

$R \Rightarrow R'$

Given layer $L = F \parallel R$ we get the following proof rule for layered architectures

$L = F \parallel R \land R \Rightarrow R'$

$\Rightarrow$

$F \Rightarrow L \otimes R'$
Composing Layers

Given two layers

\[ L \in \left[ I \rightarrow O / O' \rightarrow I' \right], \quad L' \in \left[ I' \rightarrow O'/O'' \rightarrow I'' \right] \]

where we assume that \( I \cap I'' = \emptyset \) and \( O \cap O'' = \emptyset \);
we define the layer composition

\[ L \otimes L' \in \left[ I \rightarrow O / O'' \rightarrow I'' \right] \]

yielding a layer in \( [I \rightarrow O / O'' \rightarrow I''] \).

Assume service \( L \) and \( L' \) are described as follows

\[ L = F//R \]
\[ L' = F'//R' \]

We call these two layers fitting if

\[ R \hookrightarrow F' \]

then we conclude

\[ F//R' \hookrightarrow L \otimes L' \]

proof rule for layered architectures

\[ L = F//R \land L' = F'//R' \land R \hookrightarrow F' \]

\[ \Rightarrow \]

\[ F//R' \hookrightarrow L \otimes L' \]
Layered Architecture forming a Service Stack

The service stack is specified by a set of services.

\[ F_j \in [I_j \triangleright O_j] \]

the family of export services for \(0 \leq j \leq n\).

The layers

\[ L_{j+1} \in [I_{j+1} \triangleright O_{j+1}/O_j \triangleright I_j] \]

family of layers form the stack. Given requested services

\[ R_{j+1} \in [I_j \triangleright O_j] \]

and layer specifications

\[ L_j = F_j // R_j \]

Proof condition for the stack:

\[ R_{j+1} \Rightarrow F_j \]

Service layers that fit together can be composed to service stacks.
Adding probabilistic behavior models, we can extend this approach to services writing

- Rich service specifications
  - Quality of service specification: reliability, availability, service quality
- Rich specifications of service layers
- Rich specifications of service stacks
- Rich specifications of service platforms
- Service composition
The Triad of Modeling

- **Denotational:**
  - ◊ mathematical system models (modeling behavior)
  - ◊ relating models: composition, refinement, abstraction

- **Logical – system properties:**
  - ◊ specification
  - ◊ deduction
  - ◊ verification
  - ◊ transformation

- **Notational:**
  - ◊ Graphical: diagrams
  - ◊ Tables
  - ◊ Formulas

Mathematical representations of models for understanding and getting the modeling theory right

Representations of all properties of systems (requirements, interface, functionality, architecture) in logic for specification and verification

Representations of models of systems (requirements, interface, functionality, architecture) in structured easier to grasp forms
Concluding Remarks

• The modelling framework Focus
  ◊ originally worked out for model based development
  ◊ specification
  ◊ verification
  ◊ tool support

• Tool: Autofocus 3

• Also useful for semantic foundation following the same approach
  ◊ SDL
  ◊ Bus Systems: CAN, FLEXRAY
  ◊ UML/SysML
  ◊ SOA


See: https://af3.fortiss.org/projects/autofocus3
Concluding Remarks

• Today software & systems engineering is too much orientated towards the technical architecture and solutions/implementation in the beginning

• We need a comprehensive “architectural” model-based view onto systems including requirements for dealing with complex multi-functional systems

• The models allow for
  ◊ Separation of concerns
  ◊ Separation of technical aspects from application aspects

• Technical architectures are modelled along the same theory

• Code and test cases can be generated from the models
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)