
Springer
大学计算机教育国外著名教材系列（影印版）
SOFTWARE ENGINEERING 1
ABSTRACTION AND MODELLING
软件工程
抽象与建模
卷1
Dines Bjørner 著
Software Engineering 1
CHINA-PUB.COM
清华大学出版社


SOFTWARE ENGINEERING 2
软件工程
系统与语言规约
卷2


SOFTWARE ENGINEERING 3
软件工程
领域、需求与软件设计
卷3

HELLO THERE !

Begin of Lecture 7: Last Session — Calculus II

Function Signature Discoverers and Laws

FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012

Tutorial Schedule

- **Lectures 1–2** 9:00–9:40 + 9:50–10:30

1 **Introduction** Slides 1–35

2 **Endurant Entities: Parts** Slides 36–110

- **Lectures 3–5** 11:00–11:15 + 11:20–11:45 + 11:50–12:30

3 **Endurant Entities: Materials, States** Slides 111–142

4 **Perdurant Entities: Actions and Events** Slides 143–174

5 **Perdurant Entities: Behaviours** Slides 175–285

Lunch **12:30–14:00**

- **Lecture 6–7** 14:00–14:40 + 14:50–15:30

6 **A Calculus: Analysers, Parts and Materials** Slides 286–339

$\surd$**7** **A Calculus:** **Function Signatures and Laws** **Slides 340–377**

- **Lecture 8–9** 16:00–16:40 + 16:50–17:30

8 **Domain and Interface Requirements** Slides 378–424

9 **Conclusion: Comparison to Other Work** Slides 428–460

Conclusion: What Have We Achieved Slides 425–427 + 461–472

11.3.7. ACTION_SIGNATURES

- We really should discover actions, but actually analyse function definitions.
- And we focus, in this tutorial, on just "discovering" the function signatures of these actions.
- By a function signature, to repeat, we understand
  - a functions name, say **fct**, and
  - a function type expression (**te**), say **dte**$\stackrel{\sim}{\rightarrow}$**rte** where
    - **dte** defines the type of the function's **d**efinition set
    - and **rte** defines the type of the function's image, or **r**ange set.

- We use the term ‘functions’ to cover actions, events and behaviours.
- We shall in general find that the signatures of actions, events and behaviours depend on types of more than one domain.
  - Hence the schematic index set $\{\ell_1 \widehat{\ } \langle \mathsf{t}_1 \rangle, \ell_2 \widehat{\ } \langle \mathsf{t}_2 \rangle, \ldots, \ell_n \widehat{\ } \langle \mathsf{t}_n \rangle\}$
  - is used in all action, event and behaviour discoverers.

ACTION_SIGNATURES I/II

120. The ACTION_SIGNATURES meta-function, besides narrative texts, yields
    (a) a set of auxiliary sort or concrete type definitions and
    (b) a set of action signatures each consisting of
    an action name and
    a pair of definition set and range type expressions where
    (c) the type names that occur in these type expressions
    are defined by in the domains indexed by the index set.

ACTION_SIGNATURES II/II

120 ACTION_SIGNATURES: Index $\rightarrow$ Index-**set** $\stackrel{\sim}{\rightarrow}$ (**Text**$\times$RSL)
120 ACTION_SIGNATURES$(\ell^\frown\langle \mathrm{t}\rangle)(\{\ell_1{}^\frown\langle \mathrm{t}_1\rangle,\ell_2{}^\frown\langle \mathrm{t}_2\rangle,...,\ell_n{}^\frown\langle \mathrm{t}_n\rangle\})$:
120 [narrative, possibly enumerated texts ;
120 **type** $\mathrm{t}_a,\mathrm{t}_b,...\ \mathrm{t}_c,$
120(b) **value**
120(b) $\mathrm{act}_i{:}\mathrm{te}_{i_d}\stackrel{\sim}{\rightarrow}\mathrm{te}_{i_r},\mathrm{act}_j{:}\mathrm{te}_{j_d}\stackrel{\sim}{\rightarrow}\mathrm{te}_{j_r},...,\mathrm{act}_k{:}\mathrm{te}_{k_d}\stackrel{\sim}{\rightarrow}\mathrm{te}_{k_r}$
120(c) **where:**
120(c) type names in $\mathrm{te}_{(i|j|...|k)_d}$ and in $\mathrm{te}_{(i|j|...|k)_r}$ are either
120(c) type names t_a, t_b, ... t_c or are type names defined by the
120(c) indices which are prefixes of $\ell_m{}^\frown\langle T_m\rangle$ and where T_m is
120(c) in some signature $\mathsf{act}_{i|j|...|k}$]

Example: 55 Transport Nets: Action Signatures.

- $\mathbb{ACTION\_SIGNATURES}(\langle\Delta,\mathrm{N},\mathrm{HS},\mathrm{Hs},\mathrm{H}\rangle)(\{\langle\Delta,\mathrm{N},\mathrm{LS},\mathrm{Ls},\mathrm{L}\rangle\})$:

  $\mathsf{insert\_H}: \mathsf{N} \rightarrow \mathsf{H} \stackrel{\sim}{\rightarrow} \mathsf{N}$

  $\mathsf{remove\_H}: \mathsf{N} \rightarrow \mathsf{HI} \stackrel{\sim}{\rightarrow} \mathsf{N}$

  ...

- $\mathbb{ACTION\_SIGNATURES}(\langle\Delta,\mathrm{N},\mathrm{LS},\mathrm{Ls},\mathrm{L}\rangle)(\{\langle\Delta,\mathrm{N},\mathrm{HS},\mathrm{Hs},\mathrm{H}\rangle\})$:

  $\mathsf{insert\_L}: \mathsf{N} \rightarrow \mathsf{L} \stackrel{\sim}{\rightarrow} \mathsf{N}$

  $\mathsf{remove\_L}: \mathsf{N} \rightarrow \mathsf{LI} \stackrel{\sim}{\rightarrow} \mathsf{N}$

  ...

- where $\cdots$ refer to the possibility of discovering further action signatures “rooted” in
  - $\langle\Delta,\mathrm{N},\mathrm{HS},\mathrm{Hs},\mathrm{H}\rangle$, respectively
  - $\langle\Delta,\mathrm{N},\mathrm{LS},\mathrm{Ls},\mathrm{L}\rangle$. ■

11.3.8. EVENT_SIGNATURES

EVENT_SIGNATURES I/II

121. The EVENT_SIGNATURES meta-function, besides narrative texts, yields
   (a) a set of auxiliary event sorts or concrete type definitions and
   (b) a set of event signatures each consisting of
      - an event name and
      - a pair of definition set and range type expressions

      where
   (c) the type names that occur in these type expressions are defined either in the domains indexed by the indices or by the auxiliary event sorts or types.

EVENT_SIGNATURES II/II

121 EVENT_SIGNATURES: Index $\rightarrow$ Index**-set** $\stackrel{\sim}{\rightarrow}$ (**Text**$\times$RSL)
121 EVENT_SIGNATURES($\ell \widehat{\ } \langle t \rangle$)($\{\ell_1 \widehat{\ } \langle t_1 \rangle, \ell_2 \widehat{\ } \langle t_2 \rangle, \ldots, \ell_n \widehat{\ } \langle t_n \rangle\}$):
121(a) [narrative, possibly enumerated texts omitted ;
121(a) **type** $t_a, t_b, \ldots\ t_c$,
121(b) **value**
121(b) evt_pred$_i$: $te_{d_i} \times te_{r_i} \rightarrow$ **Bool**
121(b) evt_pred$_j$: $te_{d_j} \times te_{r_j} \rightarrow$ **Bool**
121(b) ...
121(b) evt_pred$_k$: $te_{d_k} \times te_{r_k} \rightarrow$ **Bool**]

121(c) **where:** t is any of $t_a, t_b, \ldots, t_c$ or type names listed in in indices; type names of the '*d*'efinition set and '*r*'ange set type expressions te_d and te_r are type names listed in domain indices or are in $t_a, t_b, \ldots, t_c$, the auxiliary discovered event types.

Example: 56 Transport Nets: Event Signatures.
We refer to Example 34 on page 169. The omitted narrative text would, if included, as it should, be a subset of the Items 23–26 texts on Slide 167.

- $\mathbb{EVENT\_SIGNATURES}(\langle\Delta,N,LS,Ls,L\rangle)(\{\langle\Delta,N,HS,Hs,H\rangle\})$:

  value

  link_disappearance: N × N $\xrightarrow{\sim}$ **Bool**

  link_disappearance(n,n′) ≡

  $\exists\ \ell$:L · l ∈ obs_Ls(n) ⇒ pre_cond(n,ℓ) ∧ post_cond(n,ℓ,n′)

  ... [possibly further, discovered event]

  ... [signatures "rooted" in ⟨Δ,N,LS,Ls,L⟩]

- The undefined **pre_** and **post_cond**itions were "fully discovered" on Slides 169 and 171.

11.3.9. BEHAVIOUR_SIGNATURES

- We choose, in this tutorial, to model behaviours in CSP[28].
- This means that we model (synchronisation and) communication between behaviours by means of messages m of type M, CSP channels (**channel** ch:M) and CSP
  - output: ch!e [offer to deliver value of e on channel ch], and
  - input: ch? [offer to accept a value on channel ch].

[28]Other behaviour modelling languages are Petri Nets, MSCs: Message Sequence Charts, Statechart etc.

- We allow for the declaration of single channels as well as of one, two, ..., n dimensional arrays of channels with indexes ranging over channel index types
  - **type** Idx, CIdx, RIdx . . . :
  - **channel** ch:M, { ch_v[vi]:M′|vi:Idx }, { ch_m[ci,ri]:M″|ci:CIdx,ri:RIdx }, . . .

  etcetera.
- We assume some familiarity with `CSP` [Hoare85+2004] (or even `RSL/CSP` [TheSEBook1wo] [Chapter 21]).

- A behaviour usually involves two or more distinct sub-domains.

Example: 57 Vehicle Behaviour. Let us illustrate that behaviours usually involve two or more distinct sub-domains.

- A vehicle behaviour, for example, involves
  - the vehicle sub-domain,
  - the hub sub-domain (as vehicles pass through hubs),
  - the link sub-domain (as vehicles pass along links) and,
  - for the road pricing system, also the monitor sub-domain. ■

BEHAVIOUR_SIGNATURES I/II

122. The BEHAVIOUR_SIGNATURES meta-function, besides narrative texts, yields
123. It applies to a set of indices and results in a pair,
    (a) a narrative text and
    (b) a formal text:
        i. a set of one or more message types,
        ii. a set of zero, one or more channel index types,
        iii. a set of one or more channel declarations,
        iv. a set of one or more process signatures with each signature containing a behaviour name, an argument type expression, a result type expression, usually just **Unit**, and
        v. an input/output clause which refers to channels over which the signatured behaviour may interact with its environment.

BEHAVIOUR_SIGNATURES II/II

122. BEHAVIOUR_SIGNATURES: Index→ Index-**set** $\stackrel{\sim}{\rightarrow}$ (**Text**×RSL)
122. BEHAVIOUR_SIGNATURES(ℓ^⟨t⟩)({ℓ_1^⟨t_1⟩,ℓ_2^⟨t_2⟩,...,ℓ_n^⟨t_n⟩}):
123(a). [narrative, possibly enumerated texts ;
123((b))i. **type** $m = m_1 \mid m_2 \mid \ldots \mid m_\mu$, $\mu \geq 1$
123((b))ii. $i = i_1 \mid i_2 \mid \ldots \mid i_n$, $n \geq 0$
123((b))iii. **channel** c:m, {vc[x]|x:i_a}:m, {mc[x,y]|x:i_b,y:i_c}:m,...
123((b))iv. **value**
123((b))iv. bhv_1: $ate_1 \rightarrow$ $inout_1$ rte_1,
123((b))iv. ... ,
123((b))iv. bhv_m: $ate_m \rightarrow$ $inout_m$ rte_m.]
123((b))iv. **where** type expressions $atei_i$ and rte_i for all i involve at least
123((b))iv. two types t'_i, t''_j of respective indexes ℓ_i^⟨t_i⟩, ℓ_j^⟨t_j⟩,
123((b))v. **where** **Unit** may appear in either ate_i or rte_j or both.
123((b))v. **where** $inout_i$: **in** k | **out** k | **in**,**out** k
123((b))v. **where** k: c **or** vc[x] **or** {vc[x]|x:i_a•x ∈ xs} **or**
123((b))v. {mc[x,y]|x:i_b,y:i_c • x ∈ xs ∧ y ∈ ys} **or** ...

Example: 58 **Vehicle Transport: Behaviour Signatures.** We refer to Examples 35 and 36.

BEHAVIOUR_SIGNATURES($\langle\Delta$,F,VS,Vs,V$\rangle$)($\{\langle\Delta$,M$\rangle\}$):

[With each vehicle we associate behaviour with the following arguments: the vehicle identifier, the vehicle parts, and the vehicle position. The vehicle communicates with the monitor process over a vehicle to monitor array of channels, one for each vehicle ... ;

type
  VPos
channel
  $\{\mathrm{vm}[\,\mathrm{vi}\,]|\mathrm{vi}:\mathrm{VI} \cdot \mathrm{vi} \in \mathrm{vis}\}$:VPos
value
  veh: vi:VI $\to$ v:V $\to$ vp:VPos $\to$ **out** vm[vi] **Unit**]

BEHAVIOUR_SIGNATURES($\langle\Delta$,M$\rangle$)($\{\langle\Delta$,F,VS,Vs,V$\rangle\}$):

[With the monitor part we associate a behaviour with the monitor part as only argument. The monitor accepts communications from vehicle behaviours ... ;

value

mon: M $\rightarrow$ **in** $\{$vm[vi]$|$vi:VI $\cdot$ vi $\in$ vis$\}$ **Unit**]

- The "discovery" of vehicle positions into positions
  - on a link, some fraction down that link, or
  - at a hub,

  that "discovery", is left for further analysis.

We refer to Slide 192 Items 31–31(d), ■

11.4. Order of Analysis and "Discovery"

- Analysis and "discovery", that is, the "application" of
  - the analysis meta-functions and
  - the "discovery" meta-functions
- has to follow some order:
  - starts at the "root", that is with index $\langle\Delta\rangle$,
  - and proceeds with indices appending part domain type names already discovered.

11.5. Analysis and “Discovery” of “Leftovers”

- The analysis and discovery meta-functions focus on types, that is, the types
  - of abstract parts, i.e., sorts,
  - of concrete parts, i.e., concrete types,
  - of unique identifiers,
  - of mereologies, and of
  - attributes – where the latter has been largely left as sorts.

- In this tutorial we do not suggest any meta-functions for such analyses that may lead to
  - concrete types from non-part sorts, or to
  - action, event and behaviour definitions
    - say in terms of pre/post-conditions,
    - etcetera.
  - So, for the time, we suggest, as a remedy for the absence of such "helpers", good "old-fashioned" domain engineer ingenuity.

11.6. Laws of Domain Descriptions

- By a domain description law we shall understand
  - some desirable property
  - that we expect (the 'human') results of
  - the (the 'human') use of the domain description calculus
  - to satisfy.
- We may think of these laws as axioms
  - which an ideal domain description ought satisfy,
  - something that domain describers should strive for.

Notational Shorthands:

- $(f;g;h)(\Re) = h(g(f(\Re)))$
- $(f_1;f_2;\ldots;f_m)(\Re) \simeq (g_1;g_2;\ldots;g_n)(\Re)$
  means that the two "end" states are equivalent modulo appropriate renamings of types, functions, predicates, channels and behaviours.
- $[f;g;\ldots;h;\alpha]$
  stands for the Boolean value yielded by α (in state $\Re$).

11.6.1. 1st Law of Commutativity

- We make a number of assumptions:
  - the following two are well-formed indices of a domain:
    - ι': $\langle\Delta\rangle^\frown\ell'^\frown\langle\mathsf{A}\rangle$,
    - ι'': $\langle\Delta\rangle^\frown\ell''^\frown\langle\mathsf{B}\rangle$,

    where ℓ' and ℓ'' may be different or empty ($\langle\rangle$) and A and B are distinct;
  - that $\mathcal{F}$ and $\mathcal{G}$ are two, not necessarily distinct discovery functions; and
  - that the domain at ι' and at ι'' have not yet been explored.

- We wish to express,
  - as a desirable property of **domain description development**
  - that exploring domain Δ at
    - either ι' first and then ι''
    - or at ι'' first and then ι',
  - the one right after the other (hence the ";"),
  - ought yield the same partial description fragment:

124. $(\mathcal{G}(\iota'') \; ; \; (\mathcal{F}(\iota')))(\Re) \simeq (\mathcal{F}(\iota') \; ; \; (\mathcal{G}(\iota'')))(\Re)$

When a **domain description development** satisfies Law 124.,
under the above assumptions,

- then we say that the development,
- modulo type, action, event and behaviour name "assignments",
- satisfies a mild form of commutativity.

11.6.2. 2nd Law of Commutativity

- Let us assume
  - that we are exploring the sub-domain at index
  - ι: $\langle\Delta\rangle\widehat{\ }\ell\widehat{\ }\langle\mathsf{A}\rangle$.
- Whether we
  - first "discover" $\mathcal{A}$ttributes
  - and then $\mathcal{M}$ereology (including $\mathcal{U}$nique identifiers)

  or

  - first "discover" $\mathcal{M}$ereology (including $\mathcal{U}$nique identifiers)
  - and then $\mathcal{A}$ttributes

  should not matter.

- We make some abbreviations:
  - $\mathcal{A}$ stand for the $\mathbb{ATTRIBUTES}$,
  - $\mathcal{U}$ stand for the $\mathbb{UNIQUE\_IDENTIFIER}$,
  - $\mathcal{M}$ stand for the $\mathbb{MEREOLOGY}$,
  - ι for index $\langle\Delta\rangle^\frown\ell^\frown\langle\mathsf{A}\rangle$, and
  - $\iota\mathsf{s}$ for a suitable set of indices.
- Thus we wish the following law to hold:

125. $(\mathcal{A}(\iota);\mathcal{U}(\iota);\mathcal{M}(\iota)(\iota s))(\Re) \simeq$
$(\mathcal{U}(\iota);\mathcal{M}(\iota)(\iota s);\mathcal{A}(\iota))(\Re) \simeq$
$(\mathcal{U}(\iota);\mathcal{A}(\iota);\mathcal{M}(\iota)(\iota s))(\Re)$.

  - here modulo attribute and unique identifier type name renaming.

11.6.3. 3rd Law of Commutativity

- Let us again assume
  - that we are exploring the sub-domain at index
  - ι: $\langle\Delta\rangle^\frown\ell^\frown\langle\mathsf{A}\rangle$
  - where $\iota\mathsf{s}$ is a suitable set of indices.
- Whether we are
  - exploring actions, events or behaviours at that domain index
  - in that order,
  - or some other order

ought be immaterial.

- Hence with
  - $\mathcal{A}$ now standing for the $\mathbb{ACTION\_SIGNATURES}$,
  - $\mathcal{E}$ standing for the $\mathbb{EVENT\_SIGNATURES}$,
  - $\mathcal{B}$ standing for the $\mathbb{BEHAVIOUR\_SIGNATURES}$,
- discoverers, we wish the following law to hold:

126.
$$\begin{aligned}
&(\mathcal{A}(\iota)(\iota s);\mathcal{E}(\iota)(\iota s);\mathcal{B}(\iota)(\iota s))(\Re) \simeq \\
&(\mathcal{A}(\iota)(\iota s);\mathcal{B}(\iota)(\iota s);\mathcal{E}(\iota)(\iota s))(\Re) \simeq \\
&(\mathcal{E}(\iota)(\iota s);\mathcal{A}(\iota)(\iota s);\mathcal{B}(\iota)(\iota s))(\Re) \simeq \\
&(\mathcal{E}(\iota)(\iota s);\mathcal{B}(\iota)(\iota s);\mathcal{A}(\iota)(\iota s))(\Re) \simeq \\
&(\mathcal{B}(\iota)(\iota s);\mathcal{A}(\iota)(\iota s);\mathcal{E}(\iota)(\iota s))(\Re) \simeq \\
&(\mathcal{B}(\iota)(\iota s);\mathcal{E}(\iota)(\iota s);\mathcal{A}(\iota)(\iota s))(\Re).
\end{aligned}$$

  - here modulo action function, event predicate, channel, message type and behaviour (and all associated, auxiliary type) renamings.

11.6.4. 1st Law of Stability

- Re-performing
  - ◈ the same discovery function
  - ◈ over the same sub-domain,
  - ◈ that is with identical indices,
  - ◈ one or more times,

  ought not produce any new description texts.
- That is:

127. $(\mathcal{D}(\iota)(\iota\mathsf{s});\mathcal{A}\_\mathsf{and}\_\mathcal{D}\_\mathsf{seq})(\Re) \ \simeq$
$(\mathcal{D}(\iota)(\iota\mathsf{s});\mathcal{A}\_\mathsf{and}\_\mathcal{D}\_\mathsf{seq};\mathcal{D}(\iota)(\iota\mathsf{s}))(\Re)$

- where
  - ◈ $\mathcal{D}$ is any discovery function,
  - ◈ $\mathcal{A}\_\mathsf{and}\_\mathcal{D}\_\mathsf{seq}$ is any specific sequence of intermediate analyses **and d**iscoveries, and where
  - ◈ ι and $\iota\mathsf{s}$ are suitable indices, respectively sets of indices.

11.6.5. 2nd Law of Stability

- Re-performing
  - the same analysis functions
  - over the same sub-domain,
  - that is with identical indices,
  - one or more times,

  ought not produce any new analysis results.
- That is:

128. $[\mathcal{A}(\iota)] \;=\; [\mathcal{A}(\iota);\ldots;\mathcal{A}(\iota)]$

- where
  - $\mathcal{A}$ is any analysis function,
  - "…" is any sequence of intermediate analyses and discoveries, and where
  - ι is any suitable index.

11.6.6. Law of Non-interference

- When performing a discovery meta-operation, $\mathcal{D}$
  - on any index, ι, and possibly index set, $\iota\mathsf{s}$, and
  - on a repository state, $\Re$,
  - then using the $[\mathcal{D}(\iota)(\iota\mathsf{s})]$ notation
  - expresses a pair of a narrative text and some formulas, $[\mathsf{txt},\mathsf{rsl}]$,
  - whereas using the $(\mathcal{D}(\iota)(\iota\mathsf{s}))(\Re)$ notation
  - expresses a next repository state, $\Re'$.
- What is the "difference" ?
- Informally and simplifying we can say that the relation between the two expressions is:

129. $[\mathcal{D}(\iota)(\iota\mathsf{s})]$: [txt,rsl]
$(\mathcal{D}(\iota)(\iota\mathsf{s}))(\Re) = \Re'$
where $\Re' = \Re \cup \{[\mathrm{txt},\mathrm{rsl}]\}$

- We say that when 129. is satisfied
  - for any discovery meta-function $\mathcal{D}$,
  - for any indices ι and $\iota\mathsf{s}$
  - and for any repository state $\Re$,

  then the repository is not interfered with,
  - that is, *"what you see is what you get:"*

  and therefore that
  - the discovery process satisfies the law on non-interference.

11.7. Discussion

- The above is just a hint at domain development laws that we might wish orderly developments to satisfy.
- We invite the audience to suggest other laws.
- The laws of the analysis and discovery calculus
  - forms an ideal set of expectations
  - that we have of not only one domain describer
  - but from a domain describer team
  - of two or more domain describers
  - whom we expect to work, i.e., loosely collaborate,
  - based on "near"-identical domain development principles.

- These are quite some expectations.
  - But the whole point of
    - a highest-level
    - academic scientific education and
    - engineering training
  - is that one should expect commensurate development results.

- Now, since the ingenuity and creativity in the analysis and discovery process does differ between **domain developers**
  - we expect that a daily process of *"buddy checking"*,
  - where individual team members present their findings
  - and where these are discussed by the team
  - will result in adherence to the laws of the calculus.
- The laws of the analysis and discovery calculus
  - expressed some properties that we wish the repository to exhibit.

- We have deliberately abstained from "over-defining"
  - the structure of repositories and
  - the "hidden" operations (i.e., 'update', etc.)

  repositories.
- We expect further
  - research into,
  - development of,
  - possible changes to
  - and use

  of the calculus to yield such insight as to lead to
  - a firmer understanding of
  - the nature of repositories.

- In the analysis and discovery calculus
  - such as we have presented it
- we have emphasised
  - the types of parts, sorts and immediate part concrete types, and
  - the signatures of actions, events and behaviours —
  - as these predominantly featured type expressions.

- We have therefore, in this tutorial, not investigated, for example,
  - pre/post conditions of action function,
  - form of event predicates, or
  - behaviour process expressions.
- We leave that, substantially more demanding issue, for future explorative and experimental research.

End of Lecture 7: Last Session — Calculus II

Function Signature Discoverers and Laws

FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012


Software
Engineering 1


Software
Engineering 2


Software
Engineering 3

LONG BREAK