Eiffel in the real world

Thomas Beale

CTO Ocean Informatics
Chair ARB *open*EHR Foundation

Zurich

25 Nov 2010

About me…

- B Elec Eng, B Comp Sci
- 6y distributed real time systems (CMM level 4)
- 16y health and finance (CMM level 0.5)
- Main author of *open*EHR specifications
- Author of 'Archetype Definition Language' (ISO 13606-2)
- Using Eiffel since 1988

First, some psychology

~~Developers~~ We all like CONVENIENCE

- Sometimes just perceived– i.e. short-term gain over long-term value…
- May also have deep consequences
- Let's talk about convenience…

Too convenient to notice?

C_{13}

1970s

1989

“Modern Life” - Stone age

“Modern Life” - Bronze Age

“Modern Life” - Iron Age (2008)

“Modern Life” - 2010

“Modern Life” – 2012?

Past

Finance: Mandate Compliance System

Health: *open*EHR Specifications

Health: Archetype compiler for e-health

First view of Eiffel - 1988

- Leeds & Northrup (now Foxboro) SCADA real time control systems
- Motorola 68000 assembler and C
- IEEE standards-based engineering
- ~CMM 4 environment
- Ordered Eiffel 2 for Interactive Unix in 1990(?)
- Considered for adoption as reliable language / technology to replace C
  - → probably too early for the tools and libraries

Good European Health Record (GEHR)

- 1992-1995
- 3 million ecu (old style €)
- Most comprehensive work on specifications for electronic health records (EHRs) in the world to date
- Eiffel 3.x (?) on Sun workstation to:
  - Express and compile (i.e. validate) object model of interoperable EHR
  - Generate out MML (FrameMaker markup language) form of classes → integrate with main Frame document
  - → Del 19 of GEHR,
  - Influenced all later EHR standards

Finance: Mandate compliance system

- First version: 1998 – 1999
- ‘renovation’: 2006-2007
- Customer = Australia’s largest insurance company
- O(10) specialised users – fund managers
- O(100) funds, some very large: O($1b AUD)
- Each fund has a ‘mandate’ – legal def of acceptable Tx
- Mandate could be 30 rules
- ‘rule’ includes scalar and vector quantities

Design Approach

- Team: lead + 4 new devs (who did Eiffel course)
- Created a rule language, using Gobo lex/yacc tools
- Rule execution server (24x7)
- Admin tools, communicate via EiffelNet
- Rule editor GUI tool - EiffelVision
- Matisse DB
- 'Ostore' binding: DB model based on class model
  - Including mapping from Eiffel container types to native Matisse container types
    - → C# or Java programme will see same objects properly
  - Open source; available at http://www.openEHR.org SVN

Outcomes

- Development characteristics:
  - 263 Eiffel classes
  - 2000 lines of C code
  - Early version of archetypes saved customer $1m
- Deployment characteristics:
  - Eiffel + Ostore + Matisse works;
  - EiffelNet slightly arcane, but works fine
  - EiffelBuild painful to develop and maintain visual aspects, but allows 'real' app to be built
  - Performance fine

E-Health – openEHR

- *2000 -*
- *The specifications + infrastructure* (mainly UK-based work): 34,832 h, or 18.5 person years; add e.g. 50% overhead for other staff time plus infrastructure setup and maintenance and institutional overhead cost => £2.72m.
- *Open source software* (various countries): 44,000h, or 23 person years, which at the 50% overhead rate would cost £3.5m.
- *Archetypes* (Europe, Australia): 13,870h, or 7.2 person years, cost at the 50% overhead rate at *1.5 = €1.08m (converted to £0.94m)

*open*EHR specifications

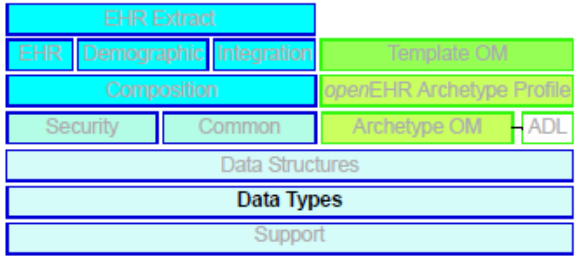
The *open*EHR Reference Model

Data Types Information Model

Editors: {T Beale, S Heard}[a], {D Kalra, D Lloyd}[b]		
Revision: 2.1.1	*Pages:* 88	*Date of issue:* 20 Nov 2008
Status: STABLE		

a. Ocean Informatics
b. Centre for Health Informatics and Multi-professional Education, University College London

Keywords: EHR, ADL, health records, modelling, constraints



The *open*EHR Foundation is an independent, non-profit community, facilitating the sharing of health records by consumers and clinicians via open-source, standards-based implementations.

Founding Chairman David Ingram, Professor of Health Informatics, CHIME, University College London

Founding Members Dr P Schloeffel, Dr S Heard, Dr D Kalra, D Lloyd, T Beale

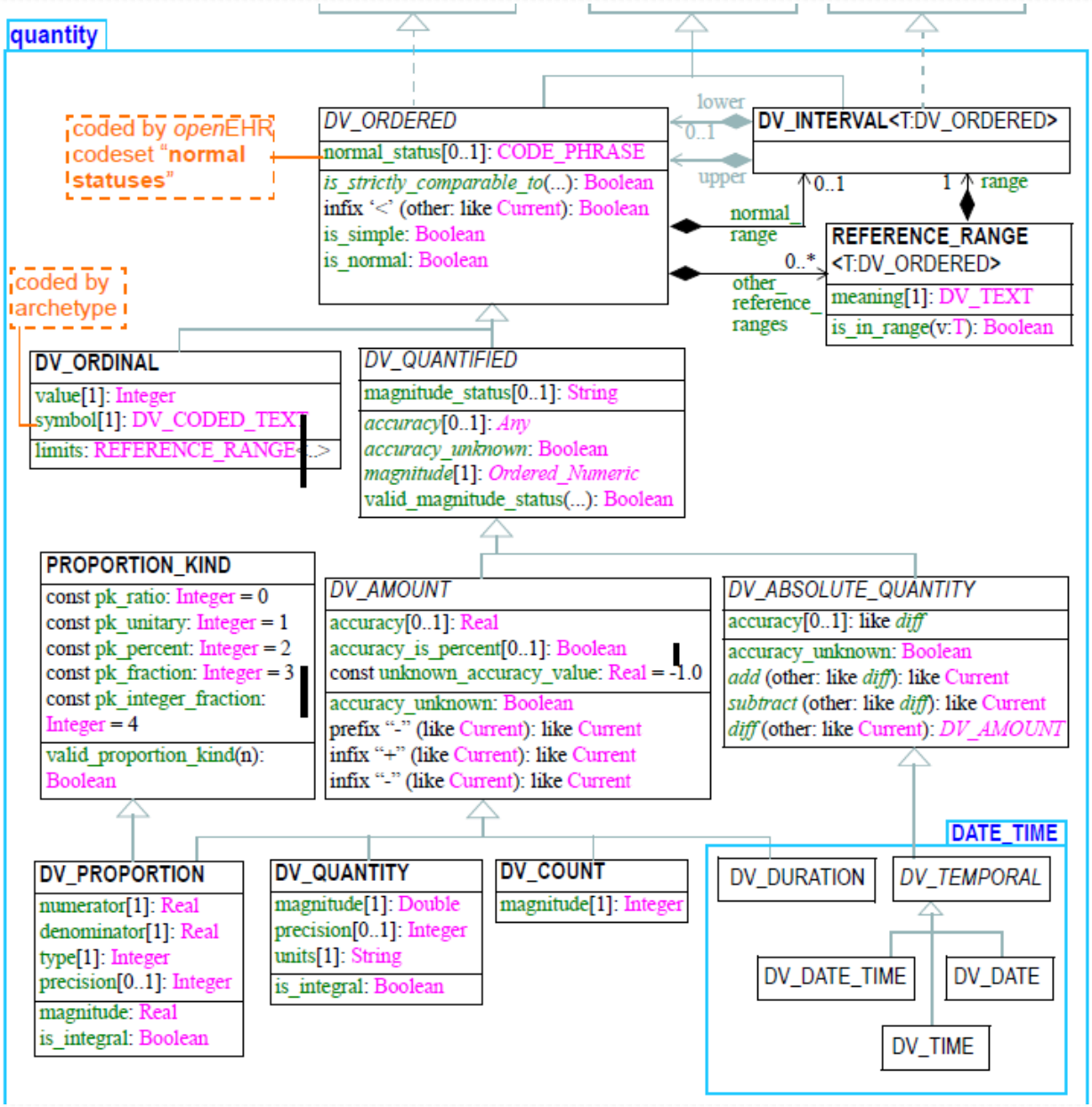
email: info@openEHR.org **web**: http://www.openEHR.org

27 specifications
~1500 pages
Eiffel-inspired

Used in:
- Australia
- Sweden
- Singapore
- Slovenia
- Slovakia
- UK
- Brazil

1 ISO standard

*open*EHR specifications (look closely)



*open*EHR specifications

6.2.4 DV_ORDINAL Class

CLASS	DV_ORDINAL
Purpose	Models rankings and scores, e.g. pain, Apgar values, etc, where there is a) implied ordering, b) no implication that the distance between each value is constant, and c) the total number of values is finite. Note that although the term 'ordinal' in mathematics means natural numbers only, here any integer is allowed, since negative and zero values are often used by medical professionals for values around a neutral point. Examples of sets of ordinal values: -3, -2, -1, 0, 1, 2, 3 -- reflex response values 0, 1, 2 -- Apgar values
Use	Used for recording any clinical datum which is customarily recorded using symbolic values. Example: the results on a urinalysis strip, e.g. {neg, trace, +, ++, +++} are used for leucocytes, protein, nitrites etc; for non-haemolysed blood {neg, trace, moderate}; for haemolysed blood {neg, trace, small, moderate, large}.
ISO 18308	STR 3.2
HL7	Quantity (QTY)
Inherit	DV_ORDERED

*open*EHR specifications

Attributes	Signature	Meaning
1..1	**value**: `Integer`	Value in ordered enumeration of values. Any integer value can be used.
1..1	**symbol**: `DV_CODED_TEXT`	Coded textual representation of this value in the enumeration, which may be strings made from "+" symbols, or other enumerations of terms such as "mild", "moderate", "severe", or even the same number series as the values, e.g. "1", "2", "3". Codes come from archetype.
Functions	**Signature**	**Meaning**
	limits: `REFERENCE_RANGE <DV_ORDINAL>`	limits of the ordinal enumeration, to allow comparison of an ordinal value to its limits.
	infix '<' (other: *like* Current): `Boolean` ***ensure*** value < other.value ***implies*** Result	True if types are the same and values compare

*open*EHR specifications

CLASS	DV_ORDINAL	
	is_strictly_comparable_to (other: ***like*** Current): `Boolean` ***ensure*** symbol.is_comparable (other.symbol) ***implies*** Result	True if symbols come from same vocabulary, assuming the vocabulary is a subset or value range, e.g. "urine:protein".
Invariants	***Symbol_exists***: symbol /= Void ***Limits_valid***: limits /= Void **and then** limits.meaning.is_equal("limits") ***Reference_range_valid***: other_reference_ranges /= Void **and then** other_reference_ranges.has(limits)	

Outcomes

- Developers love the specifications
  - Partly because of good explanatory material
  - Partly because of the contracts
- In 10 years, no-one ever complained about:
  - eiffel: STYLE
  - Generic types
  - Anchored types
  - ‘Current’
  - Contracts
  - Argumentless functions with no ‘()’
- Occasional complaint about MI
- Implemented in Java, C#, Eiffel, XSD, Python, Ruby

Archetype language and compiler

- Includes parsers for:
  - Archetype Definition Language (ADL)
    - cADL (Constraint ADL)
    - dADL (Data ADL)
    - Xpath-like assertions
- Compilation engine including
  - Validator
  - Flattener
  - Serialisers
- Object Meta-Model library
- GUI app (EiffelVision)

dADL – an XML replacement

- Half the size
- Supports
  - ‘Basic’ types, including:
    - Primitive types
    - Date, time, date_time, duration
    - List<any atomic basic type>
    - Interval<any comparable basic type>
  - Hashes, arrays, lists of complex objects
  - Shared objects, referenced by paths

dADL – basic structure

```
attr_1 = <
    attr_2 = <
        attr_3 = <leaf_value>
        attr_4 = <leaf_value>
    >
    attr_5 = <
        attr_3 = <
            attr_6 = <leaf_value>
        >
        attr_7 = <leaf_value>
    >
>
attr_8 = <>
```

```
school_schedule = <
    lesson_times = <08:30:00, 09:30:00, 10:30:00, ...>

    locations = <
        [1] = <"under the big plane tree">
        [2] = <"under the north arch">
        [3] = <"in a garden">
    >

    subjects = <
        ["philosophy:plato"] = < -- note construction of key
            name = <"philosophy">
            teacher = <"plato">
            topics = <"meta-physics", "natural science">
            weighting = <76%>
        >
        ["philosophy:kant"] = <
            name = <"philosophy">
            teacher = <"kant">
            topics = <"meaning and reason", "meta-physics", "ethics">
            weighting = <80%>
        >
        ["art"] = <
            name = <"art">
            teacher = <"goya">
            topics = <"technique", "portraiture", "satire">
            weighting = <78%>
        >
    >
```

dADL – dynamic subtyping

```
destinations = <
    ["seville"] = (TOURIST_DESTINATION) <
        profile = (DESTINATION_PROFILE) <>
        hotels = <
            ["gran sevilla"] = (HISTORIC_HOTEL) <>
            ["sofitel"] = (LUXURY_HOTEL) <>
            ["hotel real"] = (PENSION) <>
        >
        attractions = <
            ["la corrida"] = (ATTRACTION) <>
            ["Alcázar"] = (HISTORIC_SITE) <>
        >
    >
>
```

dADL – shared objects

```
destinations = <
    ["seville"] = <
        hotels = <
            ["gran sevilla"] = </hotels["gran sevilla"]>
            ["sofitel"] = </hotels["sofitel"]>
            ["hotel real"] = </hotels["hotel real"]>
        >
    >
>

bookings = <
    ["seville:0134"] = <
        customer_id = <"0134">
        period = <...>
        hotel = </hotels["sofitel"]>
    >
>

hotels = <
    ["gran sevilla"] = (HISTORIC_HOTEL) <>
    ["sofitel"] = (LUXURY_HOTEL) <>
    ["hotel real"] = (PENSION) <>
>
```

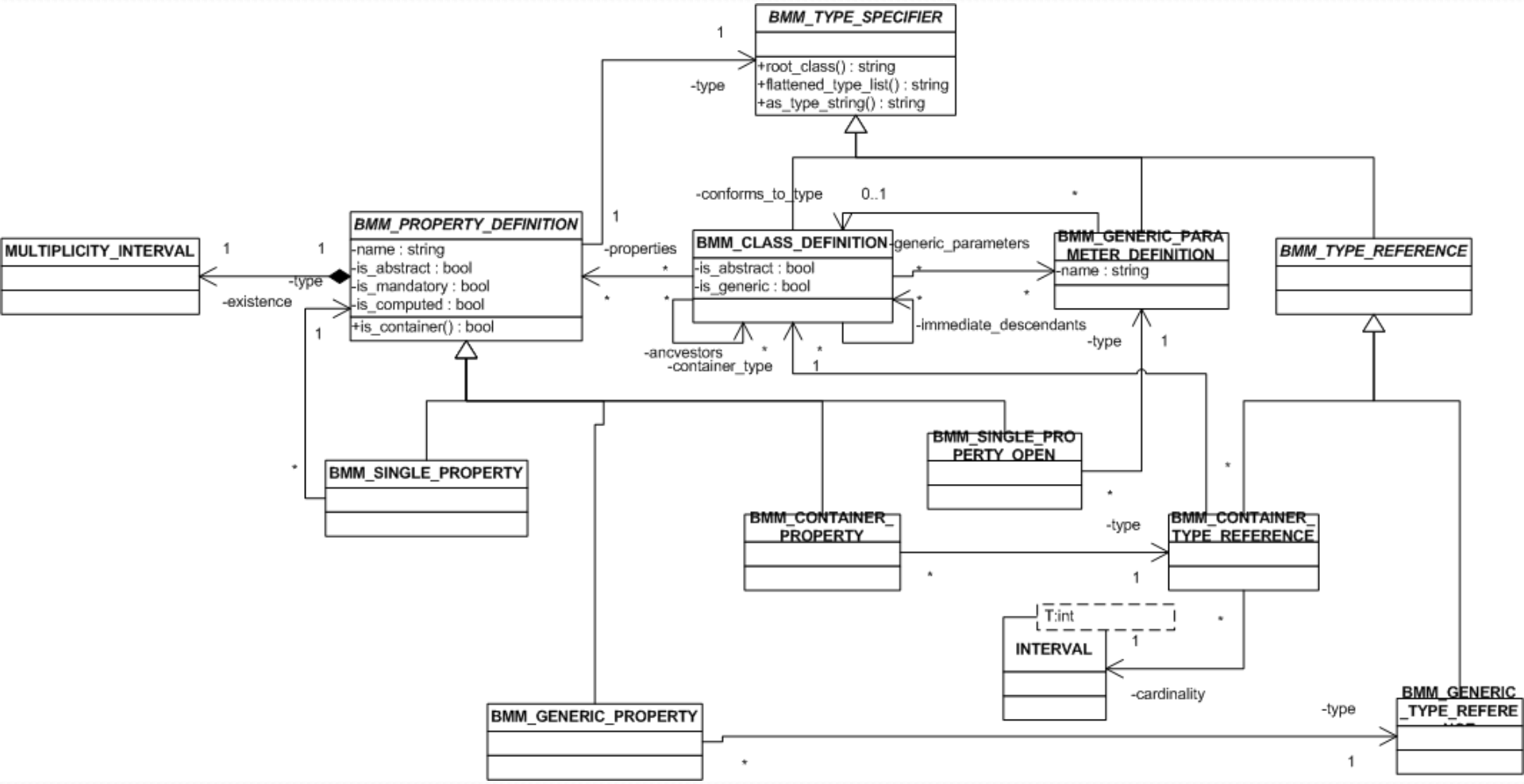
dADL – paths (Xpath-convertible)

```
/destinations["seville"]/hotels["gran sevilla"]
/destinations["seville"]/hotels["sofitel"]
/destinations["seville"]/hotels["hotel real"]

/bookings["seville:0134"]/customer_id
/bookings["seville:0134"]/period
/bookings["seville:0134"]/hotel

/hotels["sofitel"]
/hotels["hotel real"]
/hotels["gran sevilla"]
```

BMM model



Basic Meta-Model (BMM)

```
--
--------------------- rm.data_types.quantity ------------------
--

["DV_INTERVAL"] = <
    name = <"DV_INTERVAL">
    ancestors = </primitive_types["Interval"], /class_definitions["DATA_VALUE"]>
    is_generic = <True>
    generic_parameters = <
        ["T"] = <
            name = <"T">
            conforms_to_type = </class_definitions["DV_ORDERED"]>
        >
    >
    properties = <
        ["lower"] = (BMM_SINGLE_PROPERTY_OPEN) <
            name = <"lower">
            type = </class_definitions["DV_INTERVAL"]/generic_parameters["T"]>
        >
        ["upper"] = (BMM_SINGLE_PROPERTY_OPEN) <
            name = <"upper">
            type = </class_definitions["DV_INTERVAL"]/generic_parameters["T"]>
        >
    >
>

["REFERENCE_RANGE"] = <
    name = <"REFERENCE_RANGE">
    is_generic = <True>
    ancestors = </primitive_types["Any"], ...>
    generic_parameters = <
        ["T"] = <
```

Archetype Definition Language

- Archetypes are a kind of constraint model with respect to an underlying object model
  - With inbuilt semantic overloading
  - And terminology
  - And links to ontologies
- Formally understood as an F-logic query or a subset of an Powerset in an N-dimensional instance space
- Enables 'valid' object structures to be defined by domain specialists
- Is a domain-independent language that allows domain specific models to be written over an object model

cADL text

```
PERSON[at0001] ∈ {
    name ∈ {
        PERSON_NAME[at0002] ∈ {
            forenames cardinality ∈ {1..*} ∈ {/.+/}
            family_name ∈ {/.+/}
            title ∈ {"Dr", "Miss", "Mrs", "Mr", ...}
        }
    }
    addresses cardinality ∈ {1..*} ∈ {
        LOCATION_ADDRESS[at0003] ∈ {
            street_number existence ∈ {0..1} ∈ {/.+/}
            street_name ∈ {/.+/}
            locality ∈ {/.+/}
            post_code ∈ {/.+/}
            state ∈ {/.+/}
            country ∈ {/.+/}
        }
    }
}
```

```
definition
    OBSERVATION[at0000] matches {    -- ECG recording
        data matches {
            HISTORY[at0001] matches {    -- Event Series
                events cardinality matches {1..*; unordered} matches {
                    EVENT[at0002] occurrences matches {0..1} matches {   -- Any event
                        data matches {
                            ITEM_TREE[at0005] matches { -- Tree
                                items cardinality matches {0..*; unordered} matches {
                                    CLUSTER[at0006] occurrences matches {0..1} matches {     -- Global ECG Parameters
                                        items cardinality matches {1..*; unordered} matches {
                                            ELEMENT[at0013] occurrences matches {0..1} matches {     -- RR Rate
                                                value matches {
                                                    C_DV_QUANTITY <
                                                        property = <[openehr::382]>
                                                        list = <
                                                            ["1"] = <
                                                                units = <"/min">
                                                                magnitude = <|>=0.0|>
                                                                precision = <|0|>
                                                            >
                                                        >
                                                    >
                                                }
                                            }
                                            ELEMENT[at0012] occurrences matches {0..1} matches {     -- PR interval
                                                value matches {
                                                    C_DV_QUANTITY <
                                                        property = <[openehr::128]>
                                                        list = <
                                                            ["1"] = <
                                                                units = <"millisec">
                                                                magnitude = <|>=0.0|>
                                                                precision = <|0|>
                                                            >
                                                        >
                                                    >
                                                }
                                            }
```

Specialisation

Repository History Tools Help

Parse Edit openEHR-EHR-EVALUATION.problem-diagnosis.v1

ypes - openehr_rm_1.0.3

- exclusion
- family
- goal
- inpatient_summary
- (f) medical_certificate
- past_medical_history
- pregnancy
- (f) problem
  - (f) problem-diagnosis
    - (f) problem-diagnosis-sg
- reason_for_encounter
- risk
- section_summary
- simple_discharge_synopsis
- social_support_network
- substance_use_summary
- summary_plan
- travel_history
- triage

ites - openehr_rm_1.0.3

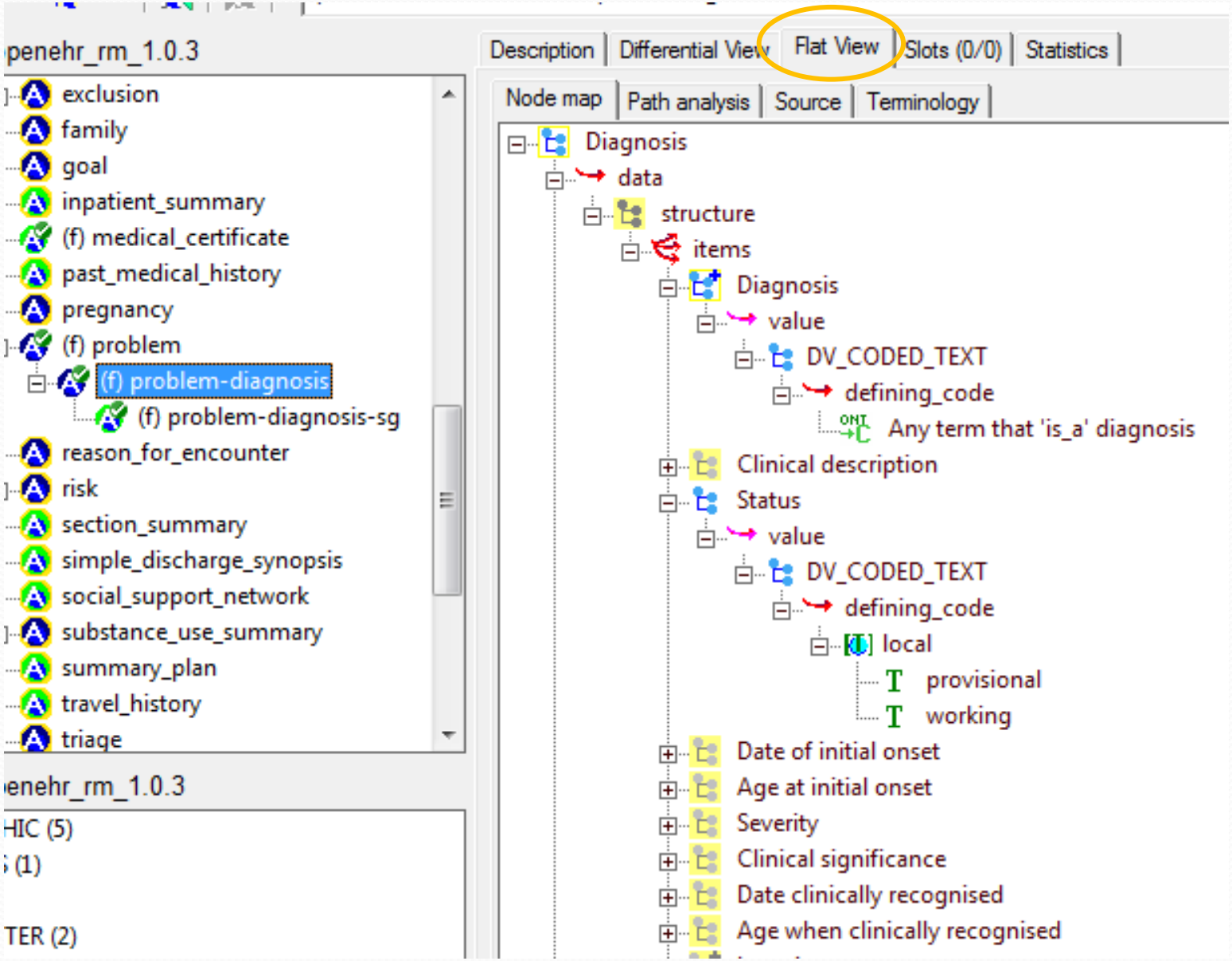
- IOGRAPHIC (5)
- ADDRESS (1)
- ITEM (2)
- CLUSTER (2)
- PARTY (1)

Description | Differential View | Flat View | Slots (0/0) | Statistics

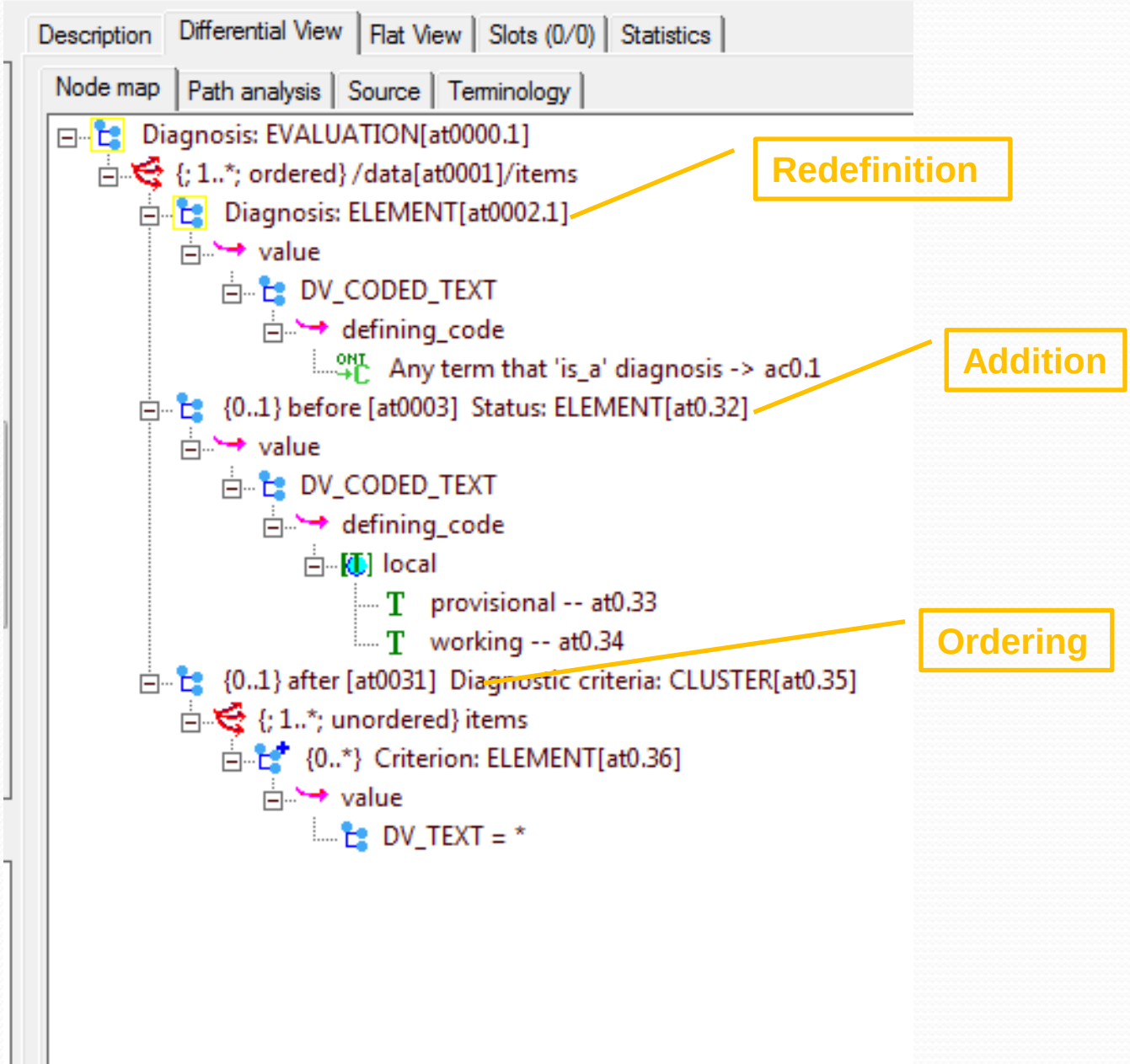
Node map | Path analysis | Source | Terminology

- Diagnosis
  - /data[at0001]/items
    - Diagnosis
      - value
        - DV_CODED_TEXT
          - defining_code
            - Any term that 'is_a' diagnosis
    - before [at0003] Status
      - value
        - DV_CODED_TEXT
          - defining_code
            - local
              - provisional
              - working
    - after [at0031] Diagnostic criteria
      - items
        - Criterion
          - value
            - DV_TEXT = *

Specialisation



Semantics



Redefinition

Addition

Ordering

Templating

openEHR-EHR-EVALUATION.problem-diagnosis-sg-t_problem_diagnosis_ds.v1

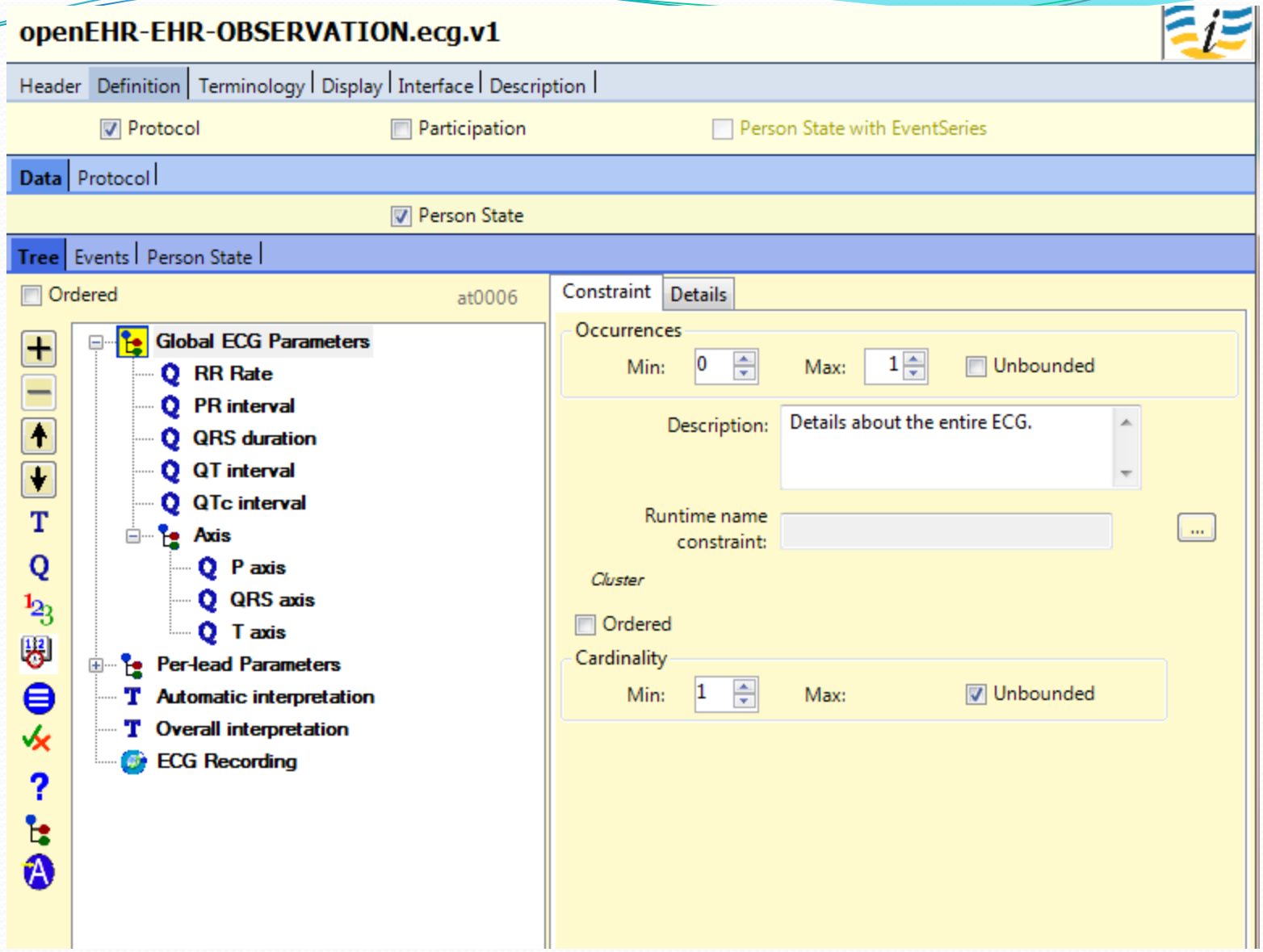
Description | Differential View | Flat View | Slots (0/0) | Statistics

Node map | Path analysis | Source | Terminology

- Singapore problem / diagnosis summary: EVALUATION[at0000.1.1.1]
  - /data[at0001]/items
    - {1} Diagnosis: ELEMENT[at0002.1.0.1] = *
    - {0} Status: ELEMENT[at0.32]
    - {0} Date of initial onset: ELEMENT[at0003]
    - {0} Age at initial onset: ELEMENT[at0004]
    - {0} Severity: ELEMENT[at0005]
    - Additional remarks: ELEMENT[at0009.0.0.1] = *
    - {0} Clinical significance: ELEMENT[at0038]
    - {0} Date clinically recognised: ELEMENT[at0010]
    - {0} Age when clinically recognised: ELEMENT[at0037]
    - {0} Location: CLUSTER[at0011]
    - {0} Aetiology: CLUSTER[at0014]
    - {0} Occurrences or exacerbations: CLUSTER[at0018]
    - {0} Related problems: CLUSTER[at0026]
    - {0} Date of resolution: ELEMENT[at0030]
    - {0} Age at resolution: ELEMENT[at0031]
    - {0} Diagnostic criteria: ELEMENT[at0.35]
    - {1} Diagnosis type: ELEMENT[at0.0.55.1] = *
  - {0} /protocol matches {*}

Mandatory

Removals



Outcomes

- Open source libraries for:
  - dADL parser
  - Data Tree (like a DOM tree)
  - Dadl ⇔ object de/serialiser
  - ADL parser & compiler
  - Basic Meta-model library
- ADL in wide use in health, including by Swedish and Australian governments
- Eiffel basic concepts like ‘flattening’ and invariants greatly eased the intellectual development

Key conclusions for IT in general

- In information and process rich domains, modelling either in the class model is out of the question
- The class model on which back-end software and databases are based can only include domain-*invariant* concepts
- Systems must be able to consume domain-variant definitions (archetypes and templates)
- Archetypes are used for gathering requirements – they are written by the domain experts, rather than IT people

Present

How to think about Development technologies

Judging development technologies...

Direct map of design

TDD
Fast compilation

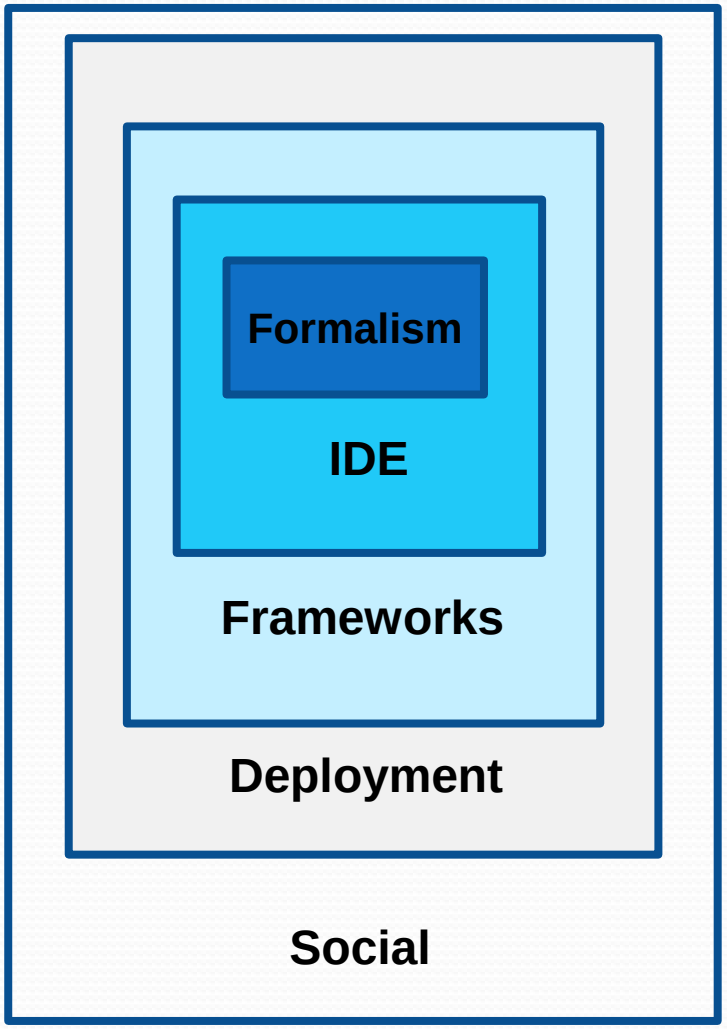
XML

Web &WS

cloud

Upgrading easy?

ideology
community



Semantic power

GUI builder
debugger

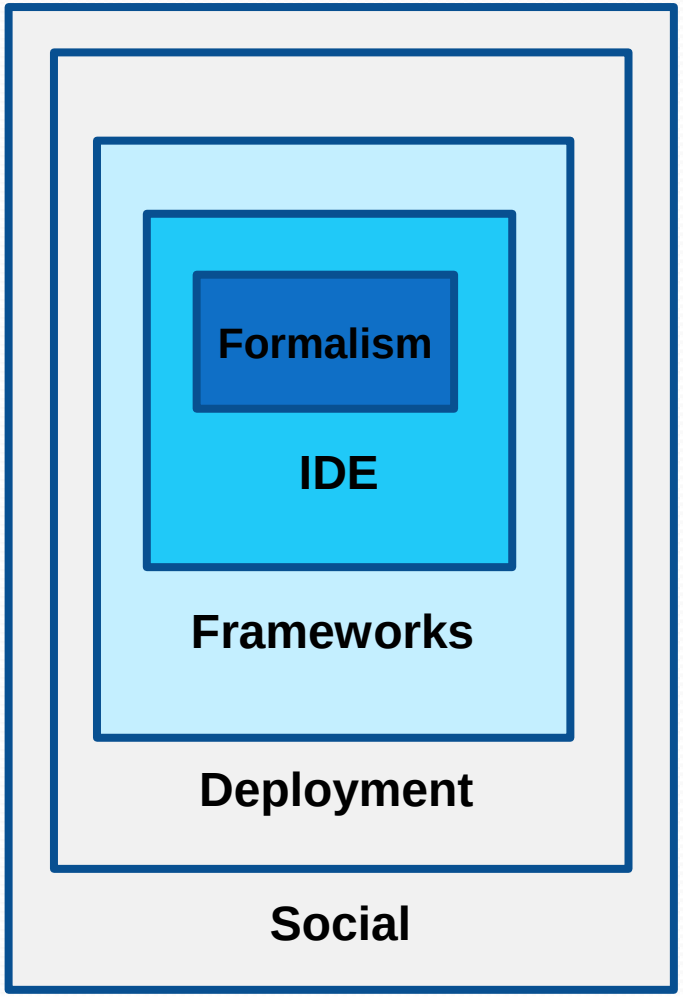
GUI
Persistence
Integ. with other langs.
interpreted

JVM
iPhone

recognition
fame

Construction Experience

The big picture



performance
availability
reliability
robustness

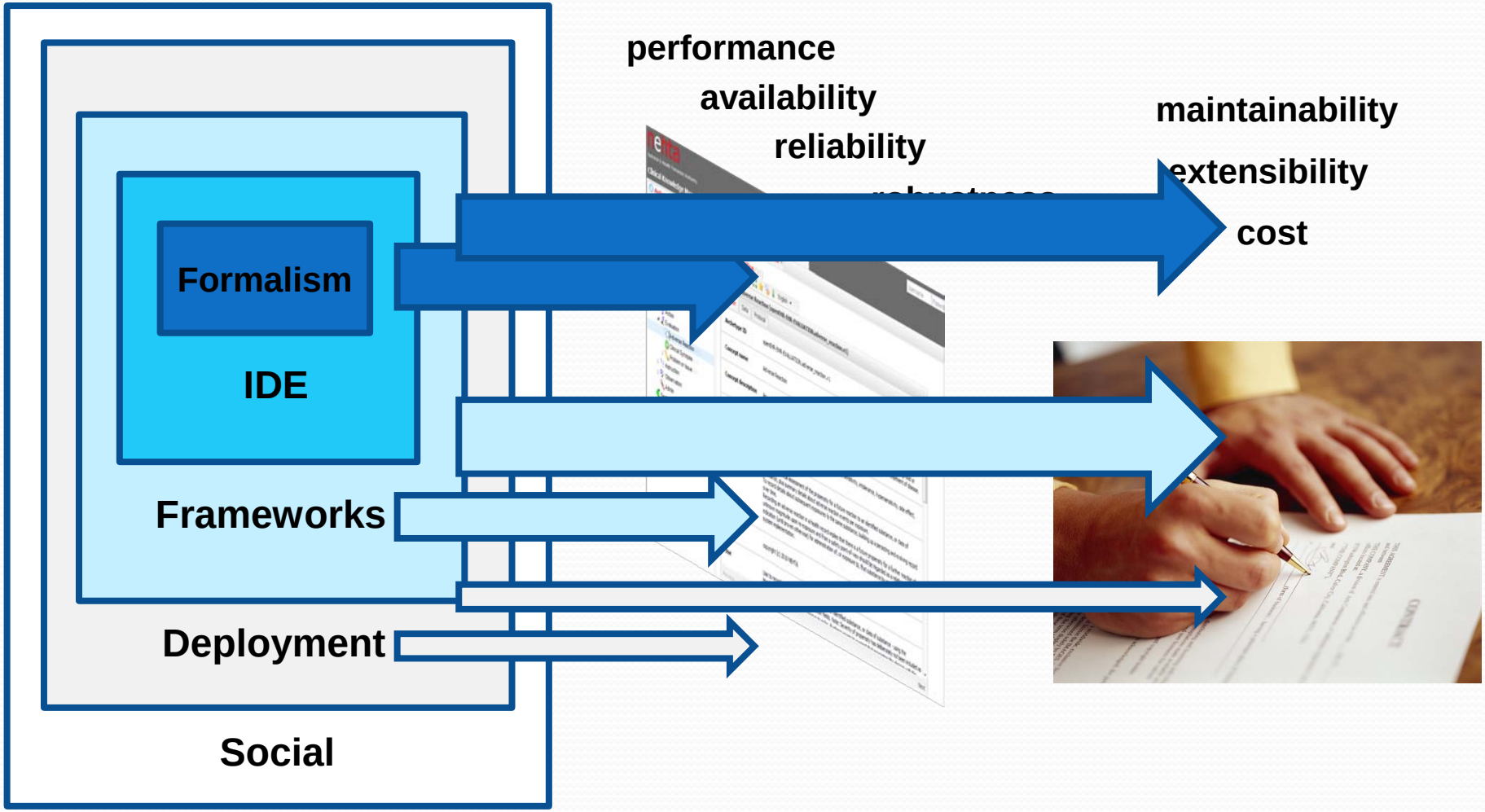
maintainability
extensibility
cost

Construction Experience

User Experience

Ownership Experience

Key value determinants

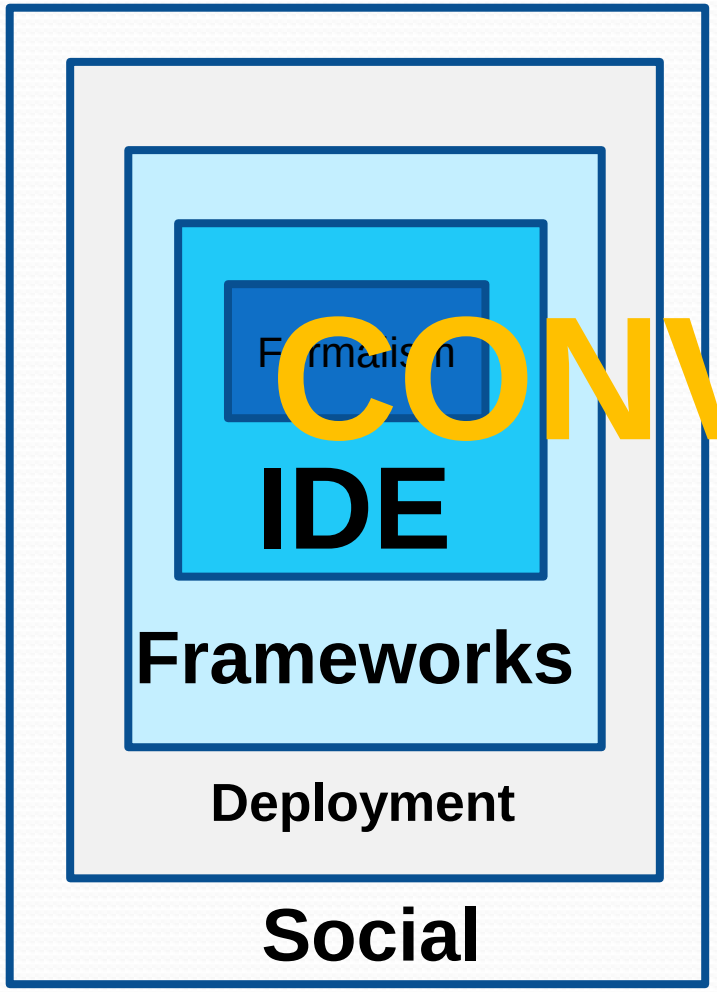


Construction Experience

User Experience

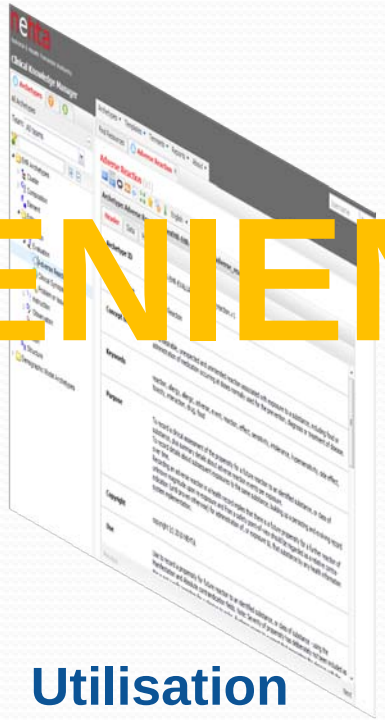
Ownership Experience

What most developers care about



Construction Experience

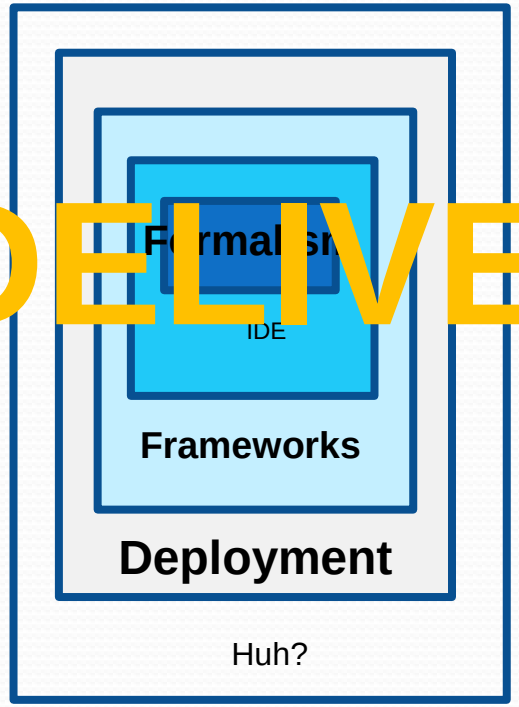
CONVENIENCE



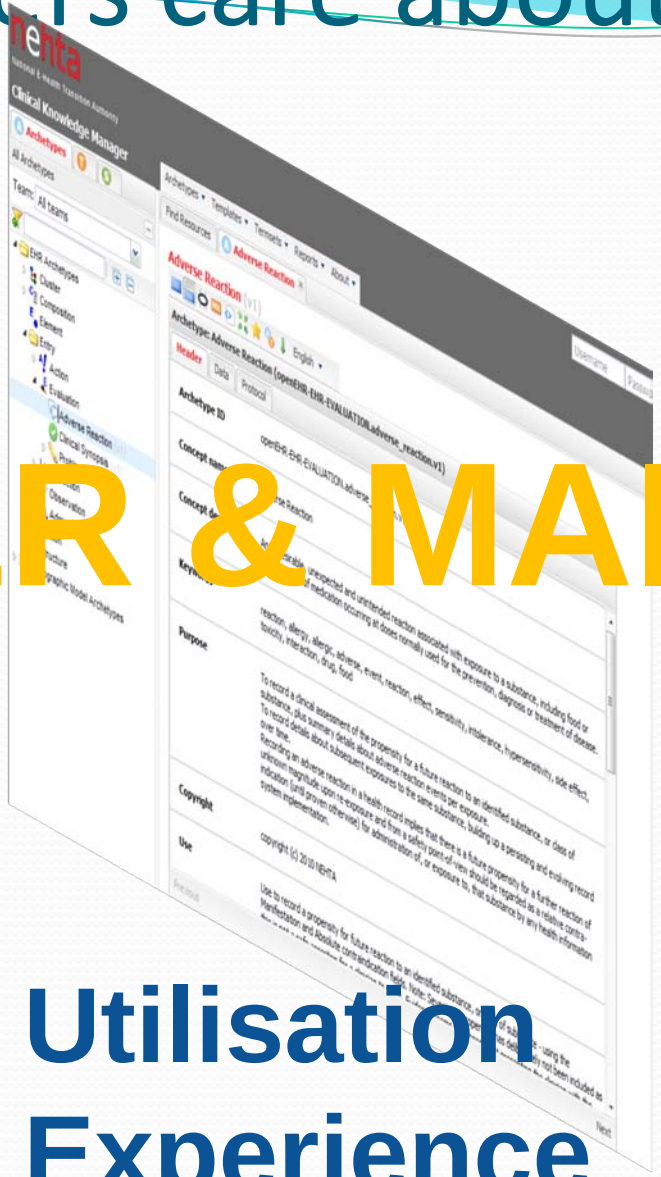
Utilisation Experience

Ownership Experience

What engineers care about



Construction Experience



Utilisation Experience

Ownership Experience

DELIVER & MAINTAIN

What business cares about

Utilisation Experience

Ownership Experience

Conclusion:

- Many developers care most about the things that have the least impact on value and quality, and most on the immediate experience
- Engineering-minded people care about value determinants particularly relating to the delivered system and its maintenance
- Business cares about the Total Cost of Ownership / Return on Investment
- Formalism, frameworks and deployment most heavily implicated in final value
- Formalism biggest determinant of ability to do good design

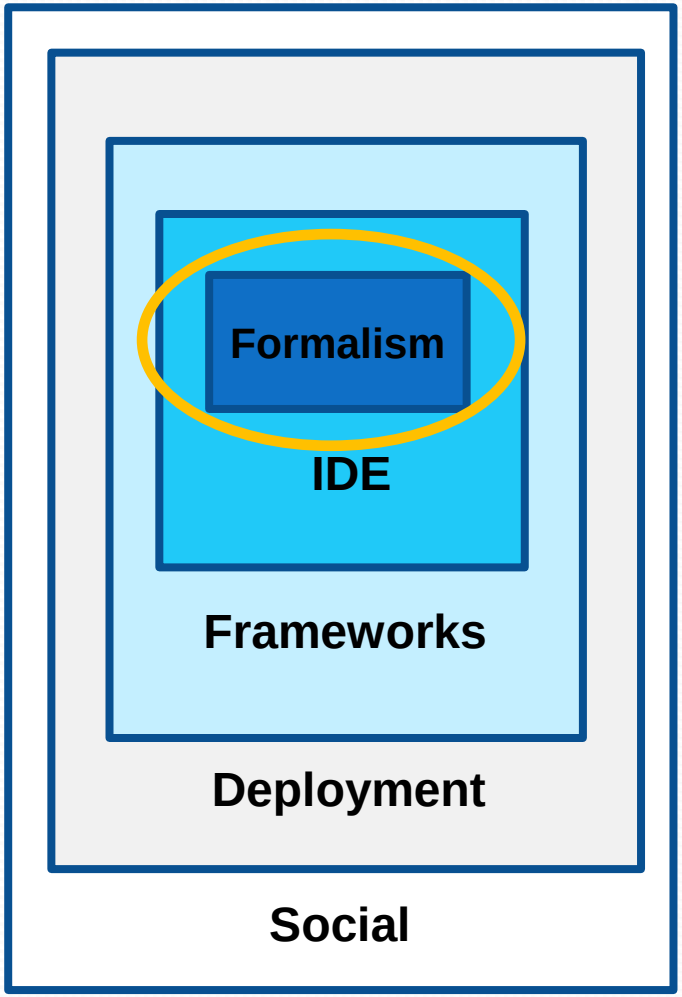
Conclusion:

- YET… in many organisations, the existing developers and the developer skills *available* on the market decide the development technology…

The attractions of Eiffel

A very formal love affair

The convenience of Eiffel

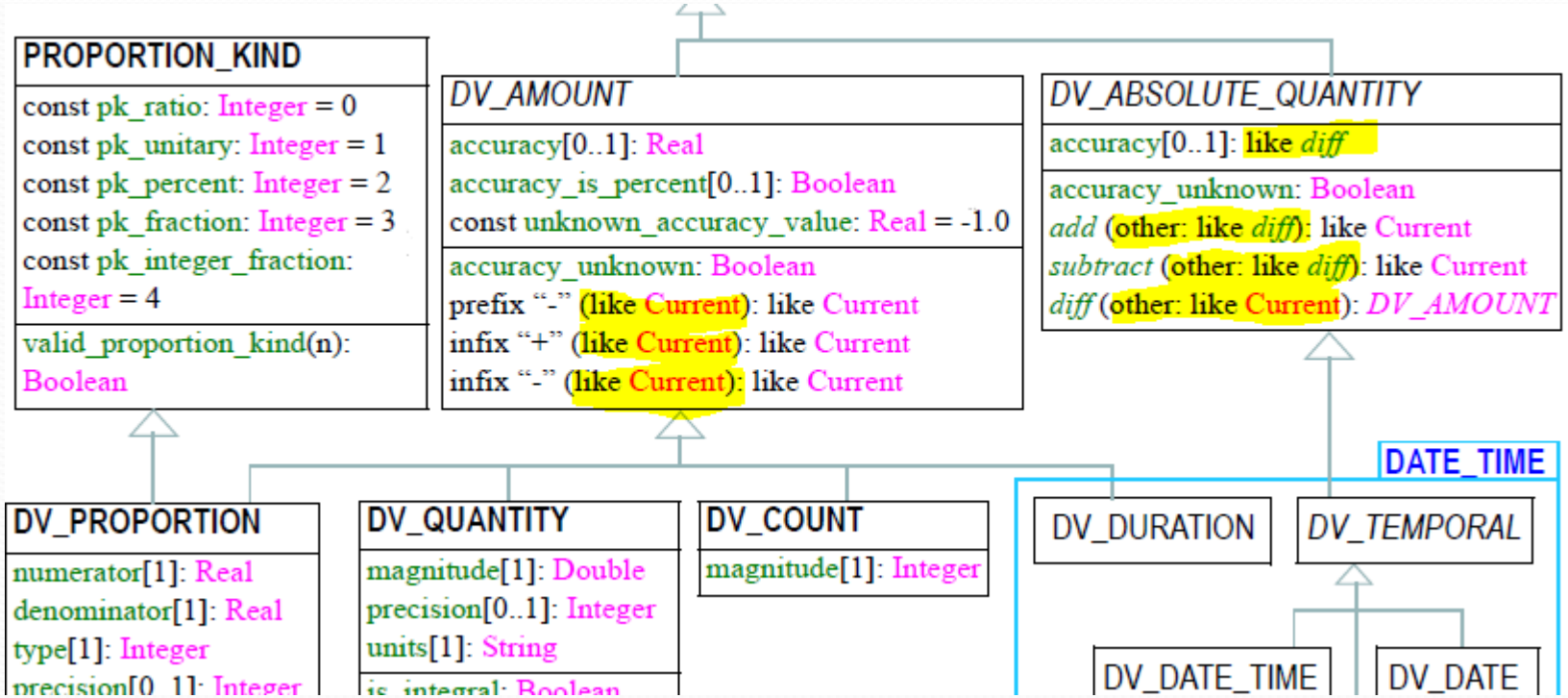


Language features we forget…

- Uniform reference:
  - {PERSON}.name, not ‘name()’
  - Client code doesn’t break if you change implementation from computed to stored
- Multiple inheritance:
  - No strange ‘extends’ v ‘implements’ rule
  - No broken memory struct mishaps
  - No fear or loathing
  - Eiffelists use it EVERYWHERE

Language features we forget…

- Anchored types:
  - Intuitive, formally correct
  - Smaller specifications



Language features we forget…

- Agents (simple form)
  - tree_iterator.do_until_surface (**agent** node_validate, **agent** node_validate_test)
  - (remember the pain before)?
- And the beautiful, but eclectic older sister:
  - tree_iterator.do_all (**agent** node_enter_action(?,?), **agent** node_exit_action(?,?))

Features we will forget soon…

- Iterator loops:
  - **across** my_list **as** ic **loop** print (ic.item) **end**
- Void safety
- Threading / SCOOP features

The last 2 may be key determiners of long-term value

Language features we never miss

- Jump statements
- Function overloading
  - No, they are not the same functions!!
- Static global functions
- Interface-mania
- Uncontrolled type casts
- And of course
  - … pointers

Things we never forget - DbC

- Ever-present
- Clarifies semantics of software
- Reduces bug diagnosis time to nearly zero
- Will probably save lives one day
- But is it really understood?

Design by Contract

- Scala and DbC:
  - On Thu, Jul 8, 2010 at 10:57 AM, David Pollak <feeder.of.the.bears@gmail.com> wrote:
    Jann,
    I was a fan of DbC until I started using Scala. One of the things that drove me out of the Ruby community was the absolute unwillingness to add DbC concepts to the language (my thought was that if optional static typing was not on the table, at least support DbC at the language level [there was a library for DbC but the syntax was not inviting.]) [REF]

Design by Contract

- From online O'Reilly book:
  - Scala doesn't provide explicit support for Design by Contract, but there are several methods in Predef that can be used for this purpose. The following example shows how to use **require** and **assume** for contract enforcement. [REF]
  - A drawback(!!!) of using these methods and **Ensuring** is that you can't disable these checks in production

Design by Contract

- From online O'Reilly book:
  - These days, the goals of Design by Contract are largely met by Test-Driven Development (TDD). However, thinking in terms of Design by Contract will complement the design benefits of TDD. If you decide to use Design by Contract in your code, consider creating a custom module that lets you disable the tests for production code.
- They are clearly unclear (!) on DbC v TDD
- Did they mention it out of guilt?!
- …many people still think DbC is a way of testing…

Design by Contract v TDD

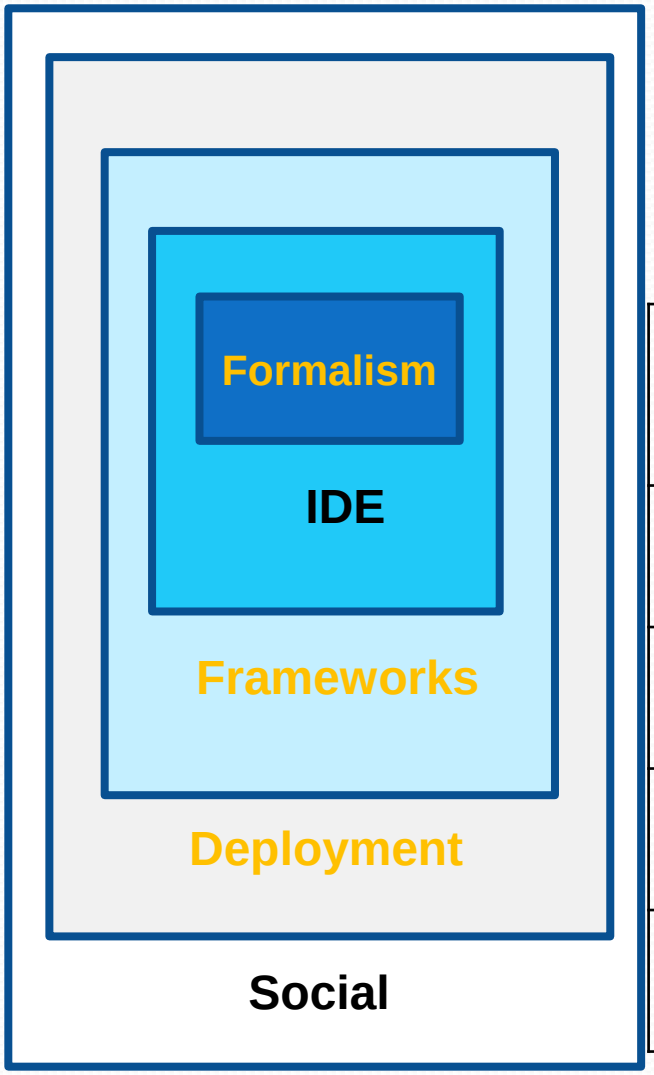
- But….
  - DbC is a mathematical specification of a valid domain (input state space) with respect to a routine of a TYPE
  - TDD is development with parallel creation of specific points in the value space (test cases) with which to test routine on an instance
- ➔ it is not a substitute,
  - Mathematically: intensional v extensional definition
  - Because client programmers don't see it, and therefore don't write better code
- ➔ we need both

Why the formalism is important

- In Eiffel, the cognitive distance between the designer's mental model and the formalism is small.
  - ➔ conceive → mental model → write 'code'
- In Java (☹ ☹), C# (☹), Python (hmm), XSD (☠), … the cognitive distance is high, and the designer spends a lot of time:
  - Fighting the formalism
  - Destroying their mental design model
  - ➔ there is no place where a clean version of their design is recorded!
- Eiffel = thinking straight into the formalism (mostly)

Technology war (just for fun)

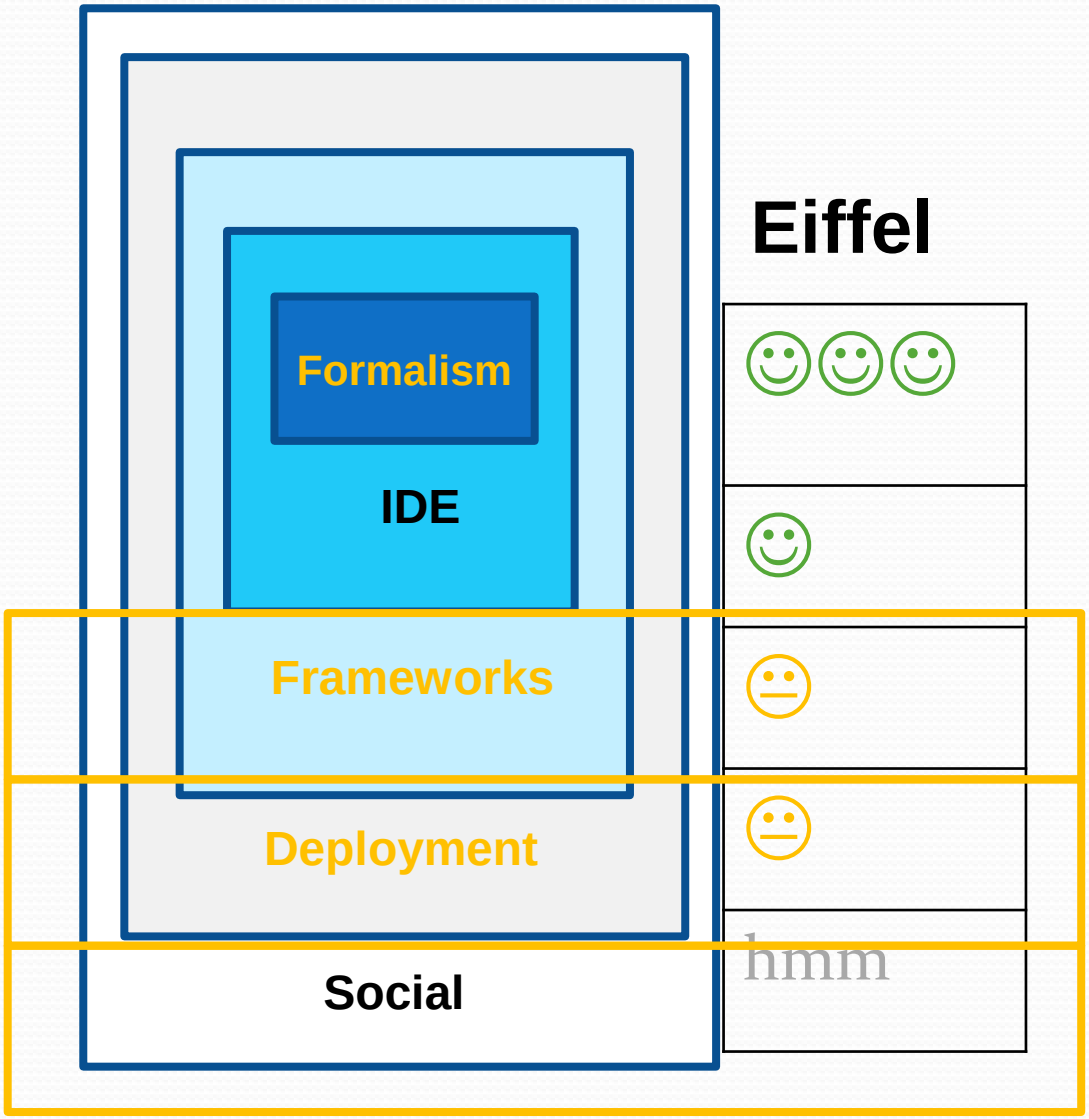
Eclipse	MS	LAMP	Eiffel
☹	😐	☠	☺☺☺
☺☺	☺☺	☹	☺
☺☺	☺☺	☺	😐
☺☺	☺	☺☺	😐
☺☺	Huh?	☺☺☺	hmm



Future

What now?

What we need to work on



Eiffel

Lacking...

☺☺☺

☺ **Visual GUI Builder**
Better pseudo-UML renderer

😐 **Easy XML Schema F/W**
Easy Web F/W

Good Java integration (JNI)

😐 **VM, interpreted mode**

hmm **Modern web 2.0 community**
It is hard to share code...

Outlook

- It is no longer about languages, it is about
  - Development technologies
    - Developer experience
    - Frameworks
  - Solution deployment capability, including upgrading
- It is not about community (in the old Usenet sense), it is about:
  - ‘Social coding’
  - Meritocracy
  - Disruption

Social aspects

- Establish a new identity and a new .org
- Create a full community web-presence
  - Website
  - Wiki with coherent, maintained documentation
  - Mailing lists
  - Coding projects:
    - Set it up like GitHub, SourceForge, CollabNet etc
    - My favourite: Atlassian: Jira, Confluence, build server, Mercurial
  - Blogs (EiffelRoom etc)
- Feed in ETHZ and other great work

Tooling

- UML has not turned out to be the killer app of development; most still use it only for drawing
- However, the 'square box' rendering is here to stay
- → improve the UML rendering, and break any rules that seem convenient, i.e. Make it a pseudo-UML tool

Deployment

- A JVM competitor is not out of the question
- Interpreted Eiffel – still a popular idea
- Consider an Eclipse-like plug-in architecture

Frameworks

- A competitor to Eclipse EMF would be easy, and industry is dying for it
- Creating and connecting to a web service needs to be easy
- Dealing with XSDs/Schematron needs to be easy

Conclusion

- Eiffel has the Language covered, and delivers well on the main value proposition
- But due to industry irrationalism, the people who 'get' this don't choose the development tools or process
- We have to think about appealing to people who want instant gratification and community…
- And give it to them!