Algorithmic check of standards for IQ dimensions

Giuseppe Primiero

FWO - Research Foundation Flanders
Centre for Logic and Philosophy of Science, Ghent University

Giuseppe.Primiero@Ugent.be
http://www.philosophy.ugent.be/giuseppeprimiero/

14 December 2012, University of Hertfordshire

Error-free Information

“Sound Information. [...] The soundness of information is usually independent of task and decision. An information consumer requires information to be error free and well represented”. [Kahn et al., 2002, p.189]

Some Questions

- What is unsound information?
- Can we characterize errors?
- Can we assess how errors affect IQ evaluation?

Tasks of this research

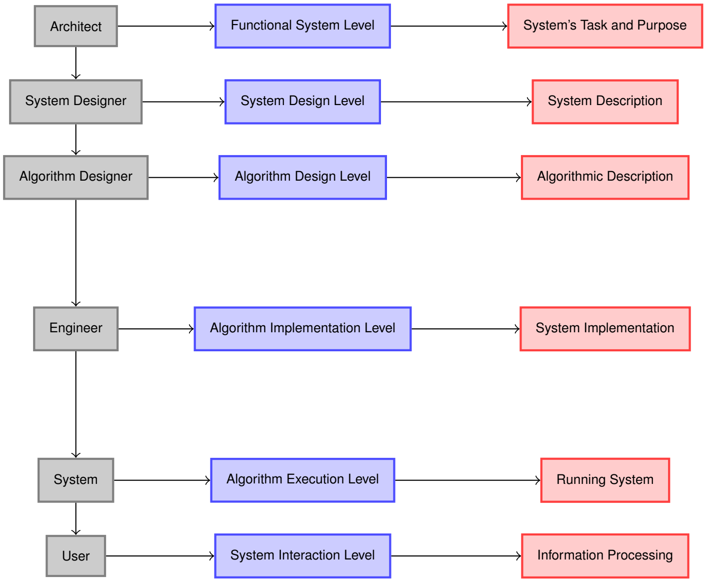
1. Set quality dimensions and their standards in a model of information processing (inspired by software production [Abran et al., 2008], [Suryn et al., 2003])
2. Design an effective way to determine if and where IQ standards fail;
3. Provide a metric by algorithmic resolution and evaluation methods;
4. Provide a formal translation of the definitions, checked in Coq.

Tasks of this research

1. Set quality dimensions and their standards in a model of information processing (inspired by software production [Abran et al., 2008], [Suryn et al., 2003])
2. Design an effective way to determine if and where IQ standards fail;
3. Provide a metric by algorithmic resolution and evaluation methods;
4. Provide a formal translation of the definitions, checked in Coq.
5. Claim: soundness as error-freeness is not level-independent

Outline

Information Flow from software production



Matching LoAs and Information Flow

LoA	AGENT	INFORMATION ACT
FSL	Architect	Semantic Purpose Definition
DSL	System Designer	Operational Representation
ADL	Algorithm Designer	Syntactical Representation
AIL	Engineer	Translation to Supported Language
AEL	System	Data Manipulation
SIL	User	Semantic Information Manipulation

A Simplified Errors Schema (based on [Primiero, 2012])

	Validity	*Correctness*	*Physical*
Conceptual	Mistake	Failure	x
Material	x	Failure/Slip	Malfunctions
Executive	x	x	Slip

Breach of Requirements

	Validity	*Correctness*	*Physical*
Conceptual	Mistake	Failure	x
Material	x	Failure/Slip	Malfunctions
Executive	x	x	Slip

Breach of Requirements (II)

- Validity requirements: the set of conditions established by the logical and semantical structure of the process defined to reach the given purpose;
- Correctness requirements: the syntactic conditions for the same process;
- Physical Rquirements: the purely contextual conditions in which the information processing is executed.

Occurrence Mode

	Validity	*Correctness*	*Physical*
Conceptual	Mistake	Failure	x
Material	x	Failure/Slip	Malfunctions
Executive	x	x	Slip

Occurrence Mode (II)

- Conceptual Mode: the aspect involved by LoAs at which the configuration and design of the system are given;
- Material Mode: the aspect involved by LoAs at which implementation of the system;
- Executive Mode: the level of successful execution and use, which can be purely accidental with respect to purpose and design.

Four Main Error Cases

1. Mistakes are errors related to the breaching of validity requirements at the functional and design levels;
2. Failures are errors related to the breaching of correctness requirements at the functional, design or implementation levels;
3. Malfunctions are errors related to the breaching of physical requirements at execution level;
4. Slips are errors related either to the breaching of correctness requirements at the implementation level; or to the breaching of physical requirements at the level of system use.

Matching Information Flow, Dimensions and Errors

AGENT	LoA	ACTION	BREACH	IQ DIMENSIONS	ERROR
Architect	FSL	Purpose Definition	Invalid	Consistency (reqs)	Mistake
			Incorrect	Accuracy (specs)	
			Incorrect	Completeness (specs)	
System Designer	DSL	Procedure Definition	Invalid	Consistency (design)	Mistake
			Incorrect	Completeness (routines)	Failure
			Incorrect	Accuracy (data)	Failure
			Incorrect	Accessibility (data)	Failure
Algorithm Designer	ADL	Algorithm selection	Invalid	Consistency (processes)	Mistake
			Invalid	Completeness (design)	Mistake
			Invalid	Relevance (design)	Mistake
			Invalid	Accuracy (design)	Mistake
Engineer	AIL	Algorithm Implementation	Incorrect	Access (data)	Failure
			Incorrect	Security (routines)	Failure
			Incorrect	Flexibility/scalability (data)	Failure
			Incorrect	Precision (I/O)	Failure
			Incorrect	Efficiency (task)	Failure
			Incorrect	Reliability (task)	Failure
			Incorrect	Sufficiency (design)	Failure
System	AEL	Execution	Unusable	Usableness	Malfunction
			Unusable	Usefulness	Malfunction
			Unusable	Accessibility (data)	Malfunction
User	SIL	Use	Unusable	Understandability	Malfunction
			Unusable	Efficiency	Malfunction
			Unusable	Precision (system)	Malfunction
			Unusable	Precision (user)	Slip
			Invalid	Relevance (purpose)	Mistake
			Incorrect	Completeness	Failure/Slip

Breaching Validity Conditions: Mistakes

- At FSL:
  - Consistency of requirements
  - Accuracy of purpose description
- At DSL:
  - Consistency of procedure definition
- At ADL:
  - Consistency of selected processes
  - Completeness of selected processes
  - Relevance of selected processes
- At SDL
  - Accuracy of selected processes
  - Accuracy of selected routines
- At SIL:
  - Relevance of system use with respect to purpose

Breaching Correctness Conditions: Failures

- At DSL
  - Completeness of selected routines
  - Accuracy of selected input data
  - Accessibility of selected input data
- At AIL
  - Accessibility of selected input data
  - Security of selected routines
  - Flexibility/Scalability of selected input data
  - Precision of Input/Output relation
  - Efficiency of task execution
  - Reliability of task execution
  - Sufficiency of task execution
- At SIL
  - Completeness of use with respect to purpose

Breaching Physical Material Conditions: Malfunctions

- At AIL
  - Accessibility of data (due to Design Failure)
  - Usability of system (due to Design Failure)
  - Usefulness of system (due to Conceptual Error)
- At SIL
  - Understandability of the system by the user (due to Design Failure)
  - Efficiency of the system (due to Design Failure)
  - Precision of the system (due to Design Failure)

Breaching Physical Executive Conditions: Slips

- At SIL
  - Precision of use by the user
  - Completeness of execution procedures by the user

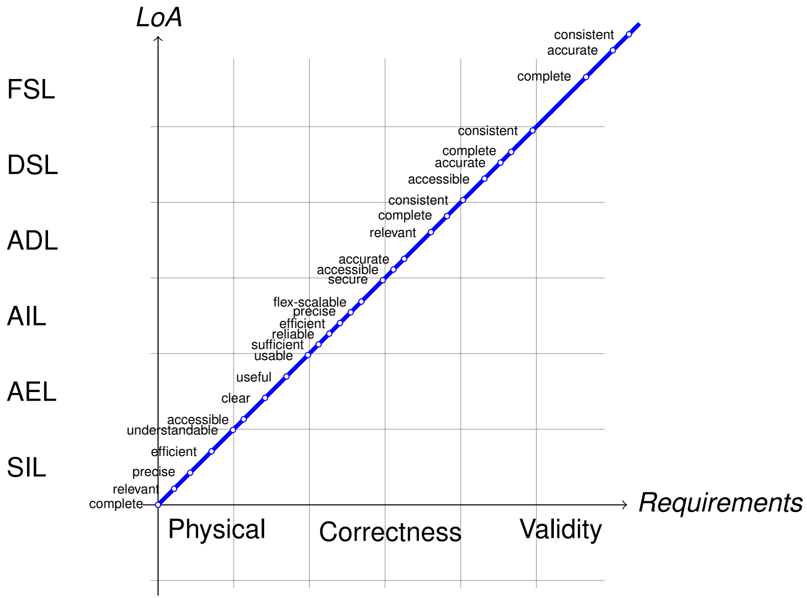
Some Remarks

- Not all the possible dimensions are included; extensions possible;
- Restriction on applicable dimensions at given LoAs:
  - e.g., *believability* holds only at SIL level (excluded for the time being)
- Some dimensions are implicit:
  - e.g., *clarity* of data could be obtained by consistent, complete and accurate design and implementation.
- Direct and detailed definitions of all dimensions and the corresponding failures are formulated formally in the code.

Defining a Metric

- Task: define an abstract metric, based on how much failures occur in the information processing, and at which LoAs do they occur
- Viewpoint: the higher the LoA at which errors occur and the more important the requirement breached, the greater the loss of quality
- for errors occurring at lower LoA, or involving less important requirements breaches, the less IQ one loses
- 2 uses: first, to establish the error-check order; second, to establish which errors are more costly in terms of information quality.

Visualizing the Model

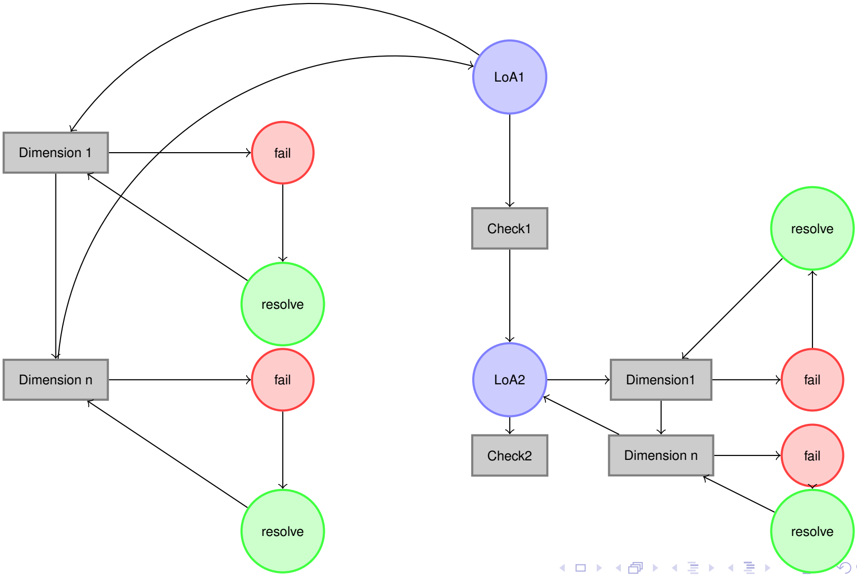


Outline

Design of the Algorithm

1. identify errors
2. resolve them
3. move back to the above level
4. proceed with the improved evaluation
5. move again down to the next dimension

Design of the Algorithm



Pseudo-Code

```
1  Start LoA1 := FSL;
2  check Dimension 1 := Consistency of
   Requirement;
3  check = yes, move to 5;
4  check = mistake, resolve and return to 2;
5  check Dimension 2 := Accuracy of
   Specifications;
6  check = yes, move to 8;
7  check = mistake, resolve and return to 5;
8  check Dimension 3 := Completeness of
   Specifications;
9  check = yes, move to 11;
10 check = mistake, resolve and return to 8;
11 Move to LoA2 := DSL;
12 ...;
n  end.
```

Pseudo-Code

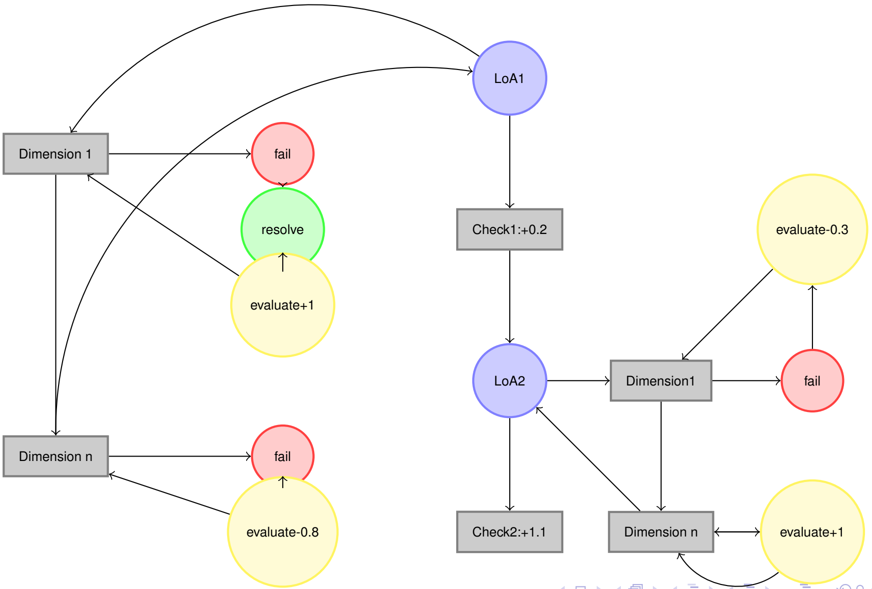
```
1  Start with System Specification;
2  check mistake in dim1 = are the requirements
   presented consistent?;
3  check = yes, move to 5;
4  check = contradictory req found, resolve and
   return to 2;
5  check mistake in dim2 = are the requirements
   presented accurate?;
6  check = yes, move to 8;
7  check = unclear req found, resolve and return
   to 5;
8  check mistake in dim3 = are the requirements
   presented complete?;
9  check = yes, move to 11;
10 check = incomplete req found, resolve and
   return to 8;
11 Move to LoA2 := DSL;
12 ...;
n  end.
```

Outline

Design of the Algorithm

1. dimension standard assessment might not require a reset of the procedure
2. only a lowering of the overall evaluation
3. allowed on some dimensions and combined with resolution for other (more crucial ones)
4. Dimensions can be listed in any desired priority order and with any desired assignment of evaluations

Design of the Algorithm



Pseudo-Code

```
      n Move to LoA4 := AIL;
    n+m ...;
  n+m+1 check failure in dim2 = are the procedures used
        secure?
  n+m+2 check = yes, move to [n+m+4];
  n+m+3 check = hackable proc found; evaluate to $-0.8$
        and move to [n+m+4];
  n+m+4 check failure in dim3 = are the routines
        effective?
  n+m+5 check = yes, return to [n+m+2] force = yes;
  n+m+6 check = no output proc, evaluate to $-1.0$ and
        return to LoA3 := ADL;
  n+m+7 ...
```

Some Remarks

1. Very scalable and highly modifiable algorithm
2. it can be strengthened or relaxed, according to required specs
3. it can be modified as to skip parameters, assign low- and high-scores
4. all depending on context and applications.

Mistakes

```
Coq < Inductive mistake : Type -> Type :=
Coq < | missing_type : mistake (forall A, Empty A)
Coq < | type_illdefined : mistake (forall t:proc, no_purpose)
Coq < | term_retype : mistake (exists A, exists t:proc,
Coq <                       In t A <-> ~ In t A).
mistake is defined
mistake_rect is defined
mistake_ind is defined
mistake_rec is defined
```

Failures

```
Coq <
Coq <
Coq < Inductive failure : Type -> Type :=
Coq < | wrong_rule : failure
(match A with match_rule => ~ A end)
Coq < | bad_rule : failure
(match A with context_rule =>  ~ A end)
Coq < | bad_address : failure
(match A with B => ~ B end)
Coq < | no_resources : failure
(match A with t => ~ t end).
Warning: pattern B is understood as a pattern variable
failure is defined
failure_rect is defined
failure_ind is defined
failure_rec is defined
```

Malfunctions

```
Coq <
Coq < Inductive malfunction : Type -> Type :=
Coq < | unusable_wrong_rule: malfunction
(exists t, match t with match_rule => ~t end)
Coq < | unusable_bad_rule : malfunction
(exists t, match t with context_rule =>  ~ t end)
Coq < | unusable_bad_address : malfunction
(exists t, match t with B => ~t end)
Coq < | unusable_no_resources : malfunction
(exists t, match t with t' => ~t end).
Warning: pattern B is understood as a pattern variable
malfunction is defined
malfunction_rect is defined
malfunction_ind is defined
malfunction_rec is defined
```

Slips

```
Coq <
Coq <
Coq < Inductive slip : Type -> Type :=
Coq < | exception_rule : slip
(~forall t t', value t -> full_eval t t')
Coq < | bad_location : slip
(~forall t1 t2 t3 t,
Coq <             full_eval t1 tm_true ->
Coq <       full_eval t2 t ->
Coq <       full_eval (tm_if t1 t2 t3) t)
Coq < | redundant_process : slip
(forall A, match A with match_rule => ~A end)
Coq < | recurrent_data : slip
(forall A, match A with t => ~t end).
slip is defined
slip_rect is defined
slip_ind is defined
slip_rec is defined
```

On Purpose (FSL)

```
Coq < Inductive invalid_purpose : Type -> Type :=
Coq < | inconsistent_req : invalid_purpose
(forall t:proc, no_purpose).
invalid_purpose is defined
invalid_purpose_rect is defined
invalid_purpose_ind is defined
invalid_purpose_rec is defined

Coq <
Coq < Inductive incorrect_purpose : Type -> Type :=
Coq < | inaccurate_req : incorrect_purpose
(match A with t => no_purpose end)
Coq < | incomplete_req : incorrect_purpose
(match A with t => forall t,
Coq < exists t1, value t -> full_eval t1 tm_true end).
incorrect_purpose is defined
incorrect_purpose_rect is defined
incorrect_purpose_ind is defined
incorrect_purpose_rec is defined
```

On Design (DSL)

```
Coq < Inductive invalid_design : Type -> Type :=
Coq < | inconsistent_design : invalid_design
(exists A, exists t:proc,  In t A <-> ~ In t A).
invalid_design is defined
invalid_design_rect is defined
invalid_design_ind is defined
invalid_design_rec is defined

Coq <
Coq < Inductive incorrect_design : Type -> Type :=
Coq < | incomplete_routine : incorrect_design
 (match A with match_rule => ~ A end)
Coq < | incomplete_routine2 : incorrect_design
 (match A with context_rule => ~ A end)
Coq < | inaccurate_data : incorrect_design
 (match A with B => ~ B end)
Coq < | inaccessible_data : incorrect_design
 (match A with t => ~ t end).
Warning: pattern B is understood as a pattern variable
incorrect_design is defined
incorrect_design_rect is defined
incorrect_design_ind is defined
```

Checking Purpose I

```
Coq <
Coq < Inductive Check_invalid_purpose : Type -> Type :=
Coq < | check_inconsistent_req : Check_invalid_purpose
Coq < (exists A:Prop, match inconsistent_req with A
=> no_purpose end).
Warning: pattern A is understood as a pattern variable
Check_invalid_purpose is defined
Check_invalid_purpose_rect is defined
Check_invalid_purpose_ind is defined
Check_invalid_purpose_rec is defined

Coq <
Coq < Inductive Check_incorrect_purpose : Type -> Type :=
Coq < | check_inaccurate_req :
 (forall A:Prop, forall t:proc, match A with t => purpose end)
Coq < Check_incorrect_purpose
(exists A:Prop, match inaccurate_req with A => no_purpose end)
```

Resolving Errors on Design I

```
Coq < Inductive Resolve_invalid_design : Type -> Type :=
Coq < | resolve_inconsistent_design :
 (exists A, exists t:proc,
Coq < match inconsistent_design with t => ~ A end) ->
Coq < Resolve_invalid_design
(exists t':proc, exists A', A' purpose).
Resolve_invalid_design is defined
Resolve_invalid_design_rect is defined
Resolve_invalid_design_ind is defined
Resolve_invalid_design_rec is defined

Coq <
Coq <
Coq < Inductive Resolve_incorrect_design : Type -> Type :=
Coq < | resolve_incomplete_routine :
 (exists A:Prop, match A with match_rule => ~ A end) ->
Coq < Resolve_incorrect_design
(exists A':Prop, match A' with match_rule => A' end)
Coq < | resolve_incomplete_routine2 :
 (exists A:Prop,match A with context_rule => ~ A end) ->
```

Resolving Errors on Design II

```
Coq < Resolve_incorrect_design
(exists A':Prop, match A' with context_rule => A' end)
Coq < | resolve_inaccurate_data :
 (exists A:Prop, exists B:Prop, match A with B => ~ B end) ->
Coq < Resolve_incorrect_design
(exists A':Prop, exists B:Prop, match A' with B => B end)
Coq < | reslve_inaccessible_data :
 (exists A:Prop, exists t:proc, match A with t => ~ t end) ->
Coq < Resolve_incorrect_design
(exists A:Prop, exists t':proc, match A with t' => t' end).
Warning: pattern B is understood as a pattern variable
Warning: pattern B is understood as a pattern variable
Resolve_incorrect_design is defined
Resolve_incorrect_design_rect is defined
Resolve_incorrect_design_ind is defined
Resolve_incorrect_design_rec is defined
```

Conclusions

- We have presented an approach to IQ assessment that relies on a negative algorithmic approach;
- Negative because it looks for errors occurring in a model of information processing;
- Algorithmic because it provides procedures to check, resolve or evaluate IQ dimensions in view of such errors;
- Currently still working on a proper translation for the resolve algorithm;
- Suggestions? Extensions? Critiques?

References I

Abran, A., Al-Qutaish, R. E., Desharnais, J.-M., and Habra, N. (2008).
Chapter 5: Iso-based models to measure software product quality,.
In Jain, B. R. K., editor, *Software Quality Measurement - Concepts and Approaches,*, pages pp. 61–96,. ICFAI University Press., Hyderabad, India.

Kahn, B. K., Strong, D. M., and Wang, R. Y. (2002).
Information quality benchmarks: product and service performance.
Commun. ACM, 45(4):184–192.

Primiero, G. (2012).
A taxonomy of errors for information systems.
Submitted.

References II

Suryn, W., Abran, A., and April, A. (2003).
Iso/iec square: The second generation of standards for software product quality.
In *Proceedings of the 7th IASTED International Conference on Software Engineering and Applications (ICSEAâ03*, pages 1–9.