Conclusion
What has all this to do with a Principle of Least Change for bidirectional transformations?

Perdita Stevens

University of Edinburgh

SSBX, July 2016

Conclusion
What has all this to do with a Principle of Least Surprise for bidirectional transformations?

Perdita Stevens

University of Edinburgh

SSBX, July 2016

Ambitious Plan

explain what everything you've heard this week has to do with Least Surprise, and lay out a research agenda for the next ten years based on that

Realistic Plan

- Why are we here? Why are bx important?
- Why do we need such a Principle? Very broadly, what do we mean by it?
- What approaches have come up this week?
- What other approaches are there?
- What next?

plus some remarks that don't fit, because I can

Setting the scene

Model-driven development isn't the only setting where bx matter – but it is the one I understand best, and I think Least Surprise is crucial there.

Why?

Model-driven development

Definition

A model is an abstract, usually graphical, representation of some aspect of a system.

Model-driven development

Definition

A model is an abstract, usually graphical, representation of some aspect of a system.

Definition

MDD is software development in which models are important.

Model-driven development

Definition

A model is an abstract, usually graphical, representation of some aspect of a system.

Definition

MDD is software development in which models are important.

But what is the problem MDD aims to solve?

Model-driven development

Definition

A model is an abstract, usually graphical, representation of some aspect of a system.

Definition

MDD is software development in which models are important.

But what is the problem MDD aims to solve?

Motivation

Manage information overload, by separating concerns and providing representations suitable for each set of decisions.

Managing information overload means...

...avoiding distracting the expert with information that isn’t relevant to them.

Managing information overload means...

...avoiding distracting the expert with information that isn't relevant to them.

In an ideal world, their model would contain all and only the information that's relevant to them

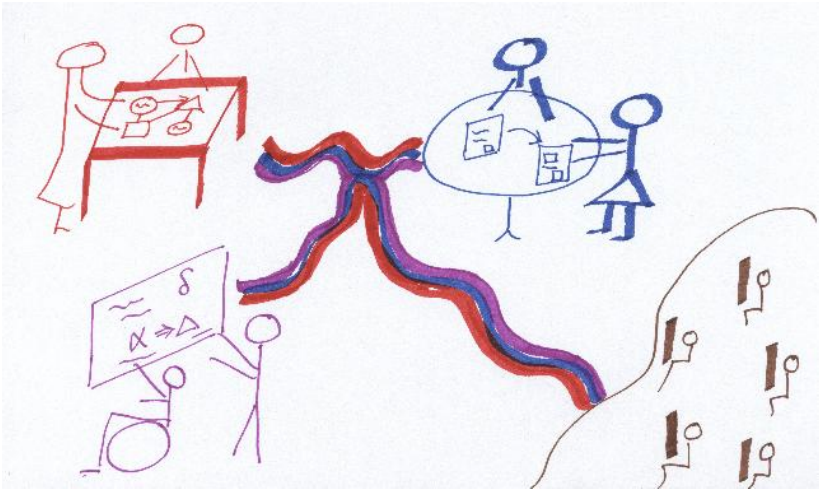
and would change only because of things they need to know anyway.

Pause for

megalomaniac

world vision

rant


δ
α ⇒ Δ

For that to work

bx will have to be trusted and trustworthy

dependable

including: reliable; maintainable; understandable; correct; available; ...

For that to work

bx will have to be trusted and trustworthy

dependable

including: reliable; maintainable; understandable; correct; available; ...

Argh! Let us bite off a small chunk and chew hard. Least Change...

Informal idea

Meertens' formulation

The action taken by the maintainer of a constraint after a violation should change no more than is needed to restore the constraint.

Extreme case is easy to formalise (and, usually, agree on):

Hippocraticness

If nothing needs to be changed, change nothing.

Next step?

Informally the next step after hippocraticness is:

don't make pointless changes

If changing just x restores consistency, and is reasonable, don't change both x and y.

Example

$M =$ UML class rectangles

$N =$ JUnit test classes

$R(m, n)$ name of m is the same as name of n, and n contains an "appropriate" set of tests for each public operation...

If one operation changes, change one test. Don't, for example, add delete or change other tests, create extra classes...

Example

$M =$ UML class rectangles

$N =$ JUnit test classes

$R(m, n)$ name of m is the same as name of n, and n contains an "appropriate" set of tests for each public operation...

If one operation changes, change one test. Don't, for example, add delete or change other tests, create extra classes...

move to the closest consistent place

Example

$M =$ UML class rectangles

$N =$ JUnit test classes

$R(m, n)$ name of m is the same as name of n, and n contains an "appropriate" set of tests for each public operation...

If one operation changes, change one test. Don't, for example, add delete or change other tests, create extra classes...

move to the closest consistent place

change should be proportionate: small change leads to small change

Who said what about Least Surprise or Least Change?

Everything in quotation marks may actually be a paraphrase – speak up if it's not what you said at all!

Mike

Constant complement as Least Change:

“You have to change the view; don’t change anything else”

Anthony

First, a different setting for Least Change:

“Suppose we have a bx, that meets its requirements. Now its requirements change. We want to be able to make a small change to the bx to meet the new requirements.”

Least change for the (LtoR privileged!) bx

$$\text{Requirements} \longleftrightarrow \text{Bx}$$

Structure

Extreme case: Richard: “we have a unidirectional MT, and then we discover we want a bx.”

Anthony

First, a different setting for Least Change:

"Suppose we have a bx, that meets its requirements. Now its requirements change. We want to be able to make a small change to the bx to meet the new requirements."

Least change for the (LtoR privileged!) bx

$$\text{Requirements} \longleftrightarrow \text{Bx}$$

Structure

Extreme case: Richard: "we have a unidirectional MT, and then we discover we want a bx."

Also: control...

Anthony

First, a different setting for Least Change:

"Suppose we have a bx, that meets its requirements. Now its requirements change. We want to be able to make a small change to the bx to meet the new requirements."

Least change for the (LtoR privileged!) bx

$$\text{Requirements} \longleftrightarrow \text{Bx}$$

Structure

Extreme case: Richard: "we have a unidirectional MT, and then we discover we want a bx."

Also: control...

who needs control?

Zhenjiang

Opening example about rectangles:

if get : $(w, h) \mapsto h$

and a 4x4 square's height is changed to 2,

what should happen to the rectangle?

Least change to what: width? aspect ratio? area?

Zhenjiang

Opening example about rectangles:

if get : $(w, h) \mapsto h$

and a 4x4 square's height is changed to 2,

what should happen to the rectangle?

Least change to what: width? aspect ratio? area?

Suggests bx developer needs control.

Zhenjiang

Opening example about rectangles:

if get : $(w, h) \mapsto h$

and a 4x4 square's height is changed to 2,

what should happen to the rectangle?

Least change to what: width? aspect ratio? area?

Suggests bx developer needs control.

Solution in the asymmetric case: define the put, derive the get: but in the symmetric case, neither direction is derivable from the other.

Aside: what first?

get first?

putback first?

think bidirectionally?

be able to add bidirectionality as an afterthought?

consistency first

Richard

Trace links as important-to-be-kept things: partly, to preserve alignment, to support least change?

Partly, for direct use, cf Anthony: “users seldom care about bx, but they care about traceability a lot, because they have to do that”.

(Here be dragons.)

Richard

Trace links as important-to-be-kept things: partly, to preserve alignment, to support least change?

Partly, for direct use, cf Anthony: "users seldom care about bx, but they care about traceability a lot, because they have to do that".

(Here be dragons.)

"Want tools that evidently implement the theory" – i.e., in practice, can dependably explain themselves?

Anthony again

pushouts as “don’t make unnecessary changes”

difficulty of finding shortest path...that’s much harder than guaranteeing no unnecessary steps...

Martin

Least surprise lurking throughout, e.g.:

Martin

Least surprise lurking throughout, e.g.:

- symmetric lenses: complements allow smooth retention of information – avoidance of surprising information loss

Martin

Least surprise lurking throughout, e.g.:

- symmetric lenses: complements allow smooth retention of information – avoidance of surprising information loss
- edit lenses: each change on one side is justified by a change that was actually, intensionally made on the other

Martin

Least surprise lurking throughout, e.g.:

- symmetric lenses: complements allow smooth retention of information – avoidance of surprising information loss
- edit lenses: each change on one side is justified by a change that was actually, intentionally made on the other

Naive approach

Let X be how much we need to change to restore the constraint.

Let Y be how much the bx actually changes.

Require $Y \leq X$.

Oh, so $Y = X$.

Now: how do we measure X and Y?

Metrics

Given m and m', need to be able to say how much changed from m to m'.

A sensible way to do this is called a metric.

Definition

A metric on M is $d : M \times M \to \mathbb{R}$
satisfying
$d(m, m') = 0$ iff $m = m'$
$d(m, m') = d(m', m)$
$d(m, m') + d(m', m'') \geq d(m, m'')$

E.g. minimal edit distance (typically).

Metric least change

Assume given:

- a bx $(R, \overrightarrow{R}, \overleftarrow{R}) : M \leftrightarrow N$;
- metrics d_M, d_N on M and N.

Then

Definition

R is metric-least if for all $m \in M$ and for all $n, n' \in N$

$$R(m, n') \Rightarrow d_N(n, n') \geq d_N(n, \overrightarrow{R}(m, n))$$

and dually.

Implementing metric least change

Macedo and Cunha implemented this approach, providing a new semantics to the syntax of QVT-R.

Given consistency relation R, and models m, n, with the task to find a new n' consistent with m:

1. set $d = 0$;
2. search exhaustively for consistent $n' \in N$ at distance d from n;
3. if there are any, present them all to the user as options;
4. else, increment d and goto 2.

Problems with metric least change

1. Scalability.
2. Usability: no easy way to ensure unique closest model, or to allow bx programmer to specify the choice.
3. Doesn't compose (even where that makes sense).
4. Inflexibility $(+/-?)$.

Can mitigate inflexibility by allowing bx programmer to choose metrics: but hmm.

Why the metric approach doesn't scale

Let S be a database, Q a PJ-query, constant size, 2 relns, so $Q(S)$ is a view.

Given: $t \in Q(S)$

Question: Is there $T \subseteq S$ s.t. $Q(S \setminus T) = Q(S) \setminus \{t\}$?

Buneman, Khanna, Tan (PODS'02): this is NP hard. (And we can't efficiently approximate, either.)

Why the metric approach doesn't scale

Let S be a database, Q a PJ-query, constant size, 2 relns, so $Q(S)$ is a view.

Given: $t \in Q(S)$

Question: Is there $T \subseteq S$ s.t. $Q(S \setminus T) = Q(S) \setminus \{t\}$?

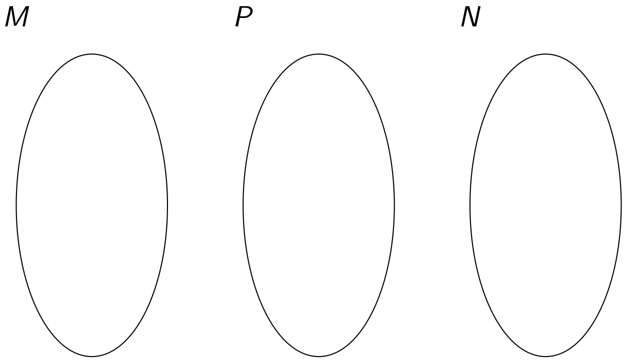
Buneman, Khanna, Tan (PODS'02): this is NP hard. (And we can't efficiently approximate, either.)

Corollary: if "models" are expressive enough to include sets of tuples (with set symmetric difference as the metric) and "consistency relations" language expressive enough to permit PJ-queries, e.g. $R_Q(m, n)$ should hold iff $Q(m) \subseteq n$, then

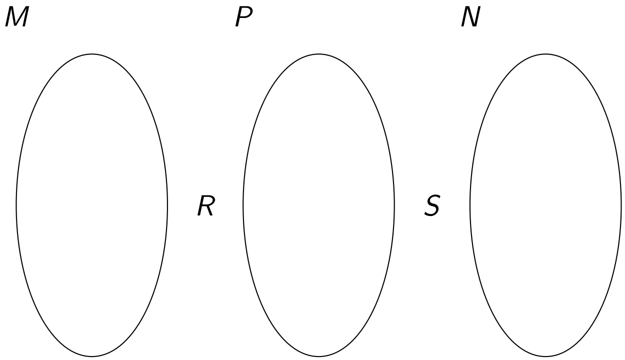
Theorem

Computing metric-least consistency restoration is NP-hard.

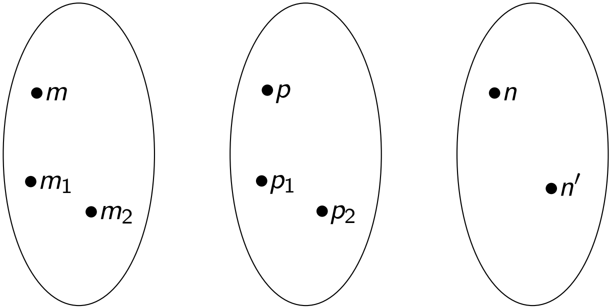
Why the metric approach doesn't compose



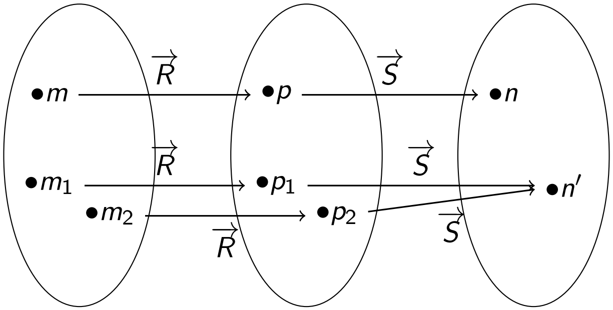
Why the metric approach doesn't compose



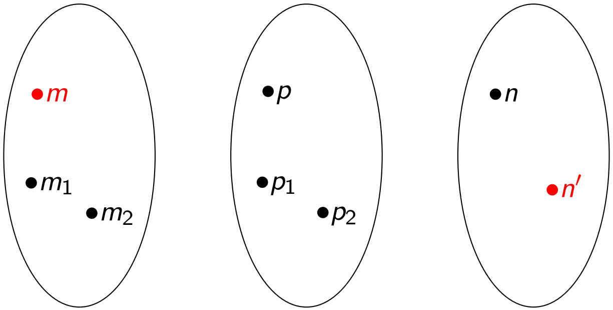
Why the metric approach doesn't compose



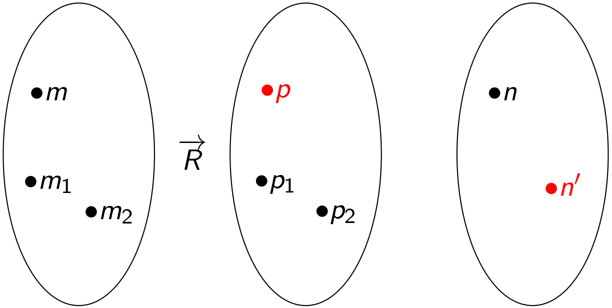
Why the metric approach doesn't compose



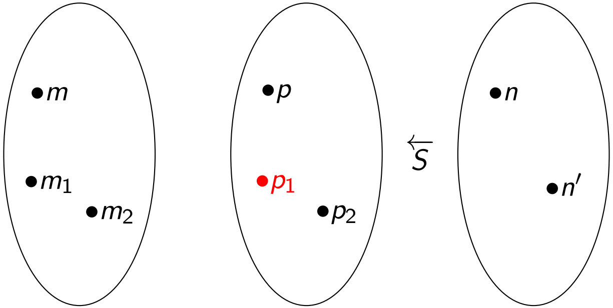
Why the metric approach doesn't compose



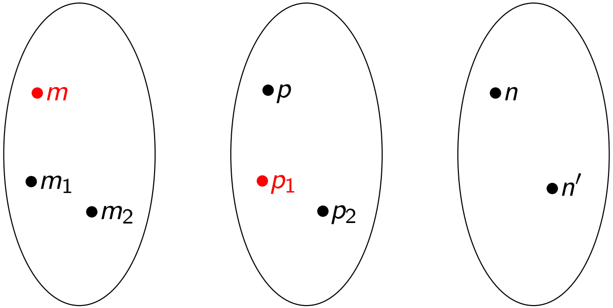
Why the metric approach doesn't compose



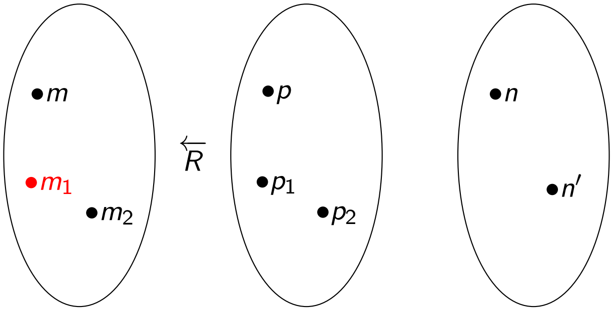
Why the metric approach doesn't compose



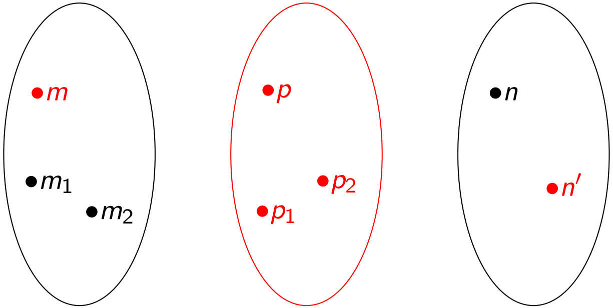
Why the metric approach doesn't compose



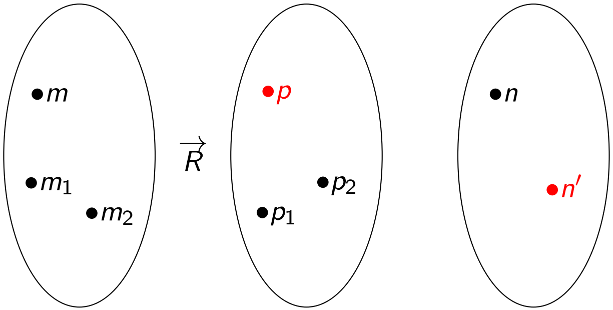
Why the metric approach doesn't compose



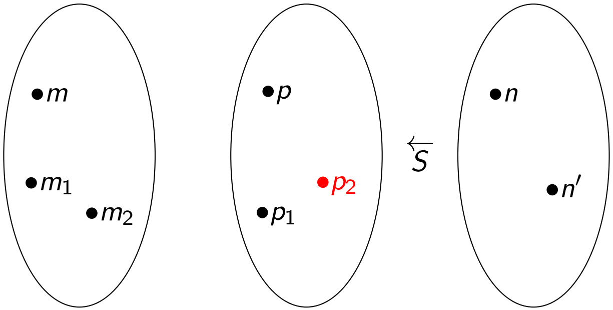
Why the metric approach doesn't compose



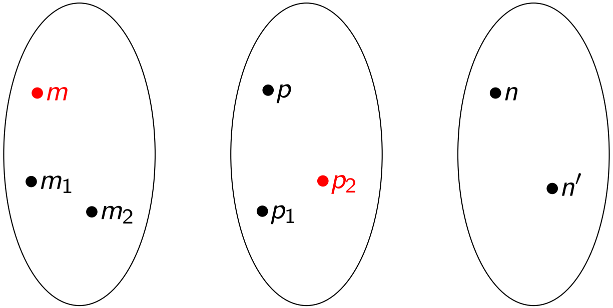
Why the metric approach doesn't compose



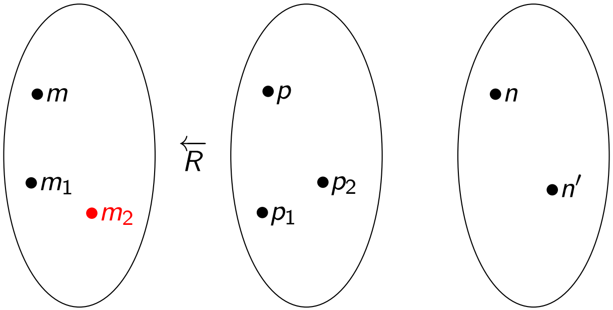
Why the metric approach doesn't compose



Why the metric approach doesn't compose



Why the metric approach doesn't compose



Example where inflexibility bites

Extent earlier example:

$M =$ UML models

$N =$ test suite (in JUnit say)

$R(m, n)$ iff every ⟪persistent⟫ class in m has a test class of the same name in n, containing an “appropriate” set of tests for each public operation...

Example where inflexibility bites

Extent earlier example:

$M =$ UML models

$N =$ test suite (in JUnit say)

$R(m, n)$ iff every ⟪persistent⟫ class in m has a test class of the same name in n, containing an "appropriate" set of tests for each public operation...

You modify the test class for a ⟪persistent⟫ class: `int` $\rightarrow$ `long` throughout.

Example where inflexibility bites

Extent earlier example:

M = UML models

N = test suite (in JUnit say)

$R(m, n)$ iff every ⟪persistent⟫ class in m has a test class of the same name in n, containing an “appropriate” set of tests for each public operation...

You modify the test class for a ⟪persistent⟫ class: `int` $\rightarrow$ `long` throughout.

Propagate back to UML model. What happens?

- `int` $\rightarrow$ `long` throughout?

Example where inflexibility bites

Extent earlier example:

M = UML models

N = test suite (in JUnit say)

$R(m, n)$ iff every ⟨⟨persistent⟩⟩ class in m has a test class of the same name in n, containing an "appropriate" set of tests for each public operation...

You modify the test class for a ⟨⟨persistent⟩⟩ class: `int` $\rightarrow$ `long` throughout.

Propagate back to UML model. What happens?

- `int` $\rightarrow$ `long` throughout?
- remove ⟨⟨persistent⟩⟩ ?

Mitigation attempt 1

Use a different metric: make removing the ⟪persistent⟫ be seen as an enormously expensive change.

But now we have a metric defined specifically for this bx.

Mitigation attempt 2

Why do we feel that removing ⟪persistent⟫ is not as cheap as it looks?

Because it disrupts the connection between the two models: breaks links.

It's a small change to the model, but a large change to the witness structure explaining their consistency?

Witness structures

Definition

A witness structure is a structure whose intention is to capture something about the relationship between the models being kept consistent.

- Relational bx: “they’re consistent because I say so”
- TGGs: “they’re consistent because this is how they are built up together using the TGG rules”
- MDD with trace links: “they’re consistent because this part of this links to that part of that”
- QVT-R game: “they’re consistent because here’s how Verifier wins a consistency game on them”
- Or even: “they’re consistent because Coq gave me this proof that they are”.

So metric least change may yet be useful

in some settings, or when its deficiencies can be mitigated as mentioned.

But it does not deserve to be seen as the sole Right Answer.

Back to the drawing board.

Principle of Least Surprise

or Least Astonishment, for user interface design, adapted to language design, etc. E.g.

Saltzer and Kaashoek, via Wikipedia

People are part of the system. The design should match the user's experience, expectations, and mental models.

Principle of Least Surprise

or Least Astonishment, for user interface design, adapted to language design, etc. E.g.

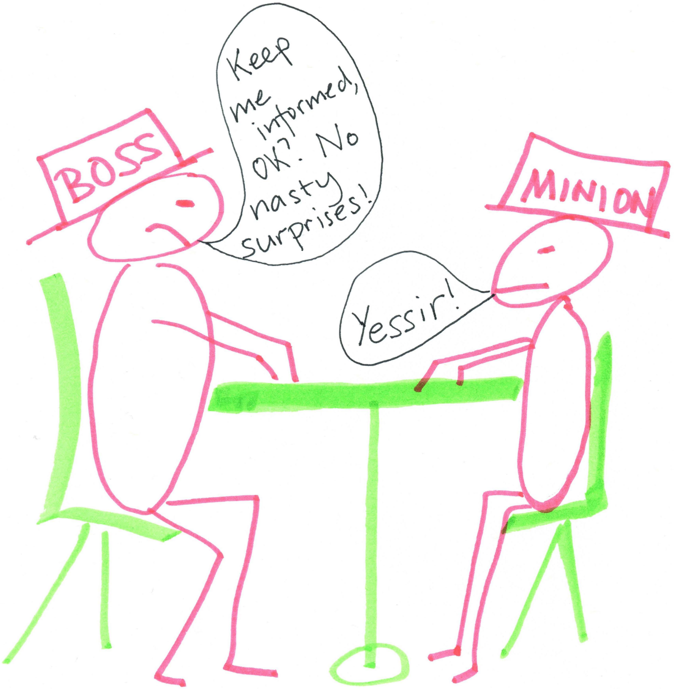
Saltzer and Kaashoek, via Wikipedia

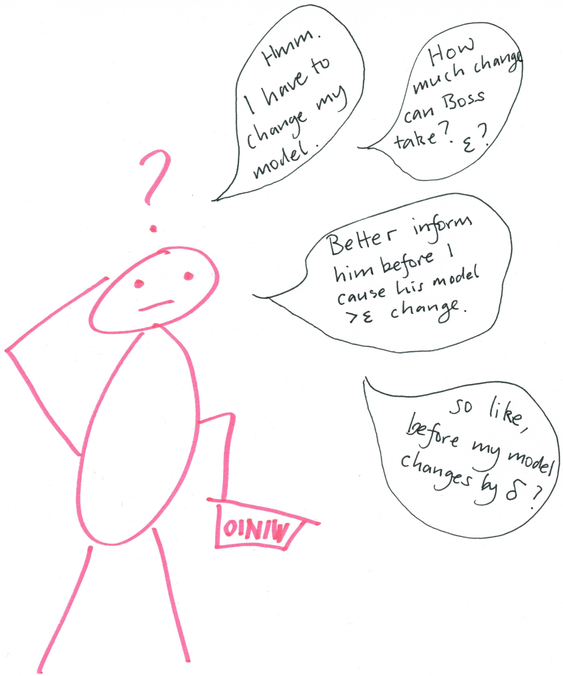
People are part of the system. The design should match the user's experience, expectations, and mental models.

The essential point is not that user should always know what will be on the next screen.

Rather, that they should seldom be surprised, i.e., taken aback.

Despite "least", we really seek reasonable, rather than optimal, behaviour.


BOSS
Keep me informed, OK?. No nasty surprises!
MINION
Yessir!


?
Hmm. I have to change my model.
How much change can Boss take? ε?
Better inform him before I cause his model >ε change.
So like, before my model changes by δ?
OINIW

Continuity

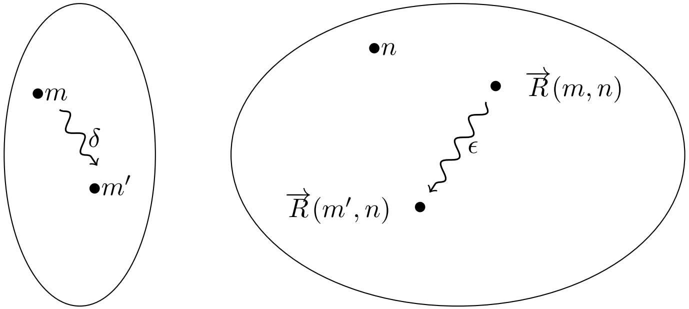
Definition

$f : S \to T$ is continuous at s iff

$$\forall \epsilon > 0 \,.\, \exists \delta > 0 \,.\, \forall s' \,.\, d_S(s, s') < \delta \Rightarrow d_T(f(s), f(s')) < \epsilon$$

We say just "f is continuous" if it is continuous at all s.

Continuity of $\overrightarrow{R}$ at (m, n)



Definition

$\overrightarrow{R}$ is continuous at (m, n) iff

$$\forall \epsilon > 0 \,.\, \exists \delta > 0 \,.\, \forall m' \,.\, d_M(m, m') < \delta \Rightarrow d_N(\overrightarrow{R}(m, n), \overrightarrow{R}(m', n)) < \epsilon$$

i.e. iff $\overrightarrow{R}(\_, n) : M \to N$ is continuous at m.

Where do we want continuity?

Well, at the points where we want guarantees... e.g.

- Everywhere: strong continuity
- Only at consistent (m, n): weak continuity
- Something else? Where we can get it? On "good" subspace pair?

In the history ignorant (vwb, PUTPUT) case, strong $=$ weak.

$+$ Strongly (rsp. weakly) continuous bx compose, where composition makes sense.

The problem with continuity

– Every function is continuous at an isolated point :-(

The problem with continuity

– Every function is continuous at an isolated point :-(

Variants with better overall behaviour? See Bx15 paper. Hölder continuity promising.

Hölder continuity

idea: systematically bound the amount of change in the result of a function, in terms of the amount of change in the argument to the function.

Hölder continuity

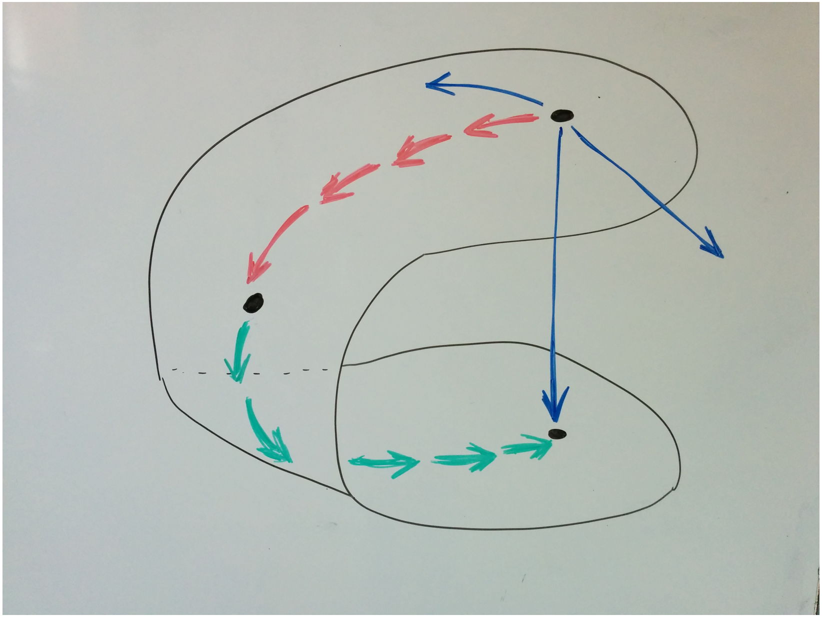
idea: systematically bound the amount of change in the result of a function, in terms of the amount of change in the argument to the function.

Definition

$\overrightarrow{R}$ is Hölder continuous (with respect to C, α) at (m, n) i ff

$$\forall m' \,.\, d_N(\overrightarrow{R}(m, n), \overrightarrow{R}(m', n)) \leq C d_M(m, m')^\alpha$$

Prediction: way too strong to expect it everywhere in realistic bx, but nevertheless, Hölder continuity may be worthwhile to think about.



Problem

as usual, we want more than we can have

Our least surprise properties are too strong to expect them to hold on realistic bx.

Problem

as usual, we want more than we can have

Our least surprise properties are too strong to expect them to hold on realistic bx.

Identify where they do hold, and build a

Tool that Goes Beep

The Tool that Goes Beep

The tool knows its own limitations.

The Tool that Goes Beep

The tool knows its own limitations.

From some states of the world – e.g. pairs (m, n) of models – it knows how to restore consistency in an unsurprising way.

If asked to, it will do so silently.

The Tool that Goes Beep

The tool knows its own limitations.

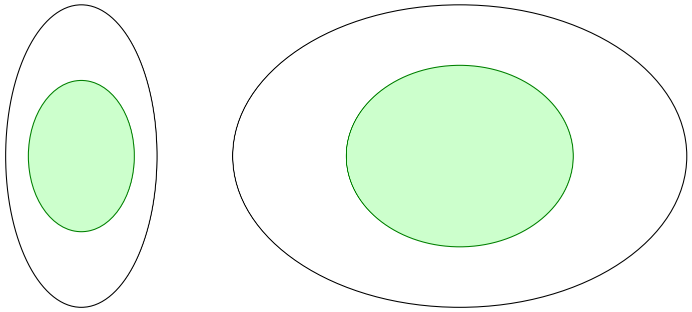
From some states of the world – e.g. pairs (m, n) of models – it knows how to restore consistency in an unsurprising way.

If asked to, it will do so silently.

From other states of the world, it doesn't, or isn't confident, so it will beep.

The user may want to check, perhaps even amend, what the tool did – or the tool may have given up and done nothing.

Subspace pairs as good states of the world



Why use subspace pairs?

The states of the world from which the tool knows a good thing to do needn't form an ssp – why is it useful to think about ssps?

- Bx restrict to bx on ssps:

Why use subspace pairs?

The states of the world from which the tool knows a good thing to do needn't form an ssp – why is it useful to think about ssps?

- Bx restrict to bx on ssps:
- may behave better (less surprisingly) on an ssp than on the whole space

Why use subspace pairs?

The states of the world from which the tool knows a good thing to do needn't form an ssp – why is it useful to think about ssps?

- Bx restrict to bx on ssps:
- may behave better (less surprisingly) on an ssp than on the whole space
- no surprising exit from an ssp: exit only when the user causes it.

Why use subspace pairs?

The states of the world from which the tool knows a good thing to do needn't form an ssp – why is it useful to think about ssps?

- Bx restrict to bx on ssps:
- may behave better (less surprisingly) on an ssp than on the whole space
- no surprising exit from an ssp: exit only when the user causes it.
- so in particular, if your organisation can agree to stay in an ssp where the bx is good enough, your beeping tool need never beep.

Why use subspace pairs?

The states of the world from which the tool knows a good thing to do needn't form an ssp – why is it useful to think about ssps?

- Bx restrict to bx on ssps:
- may behave better (less surprisingly) on an ssp than on the whole space
- no surprising exit from an ssp: exit only when the user causes it.
- so in particular, if your organisation can agree to stay in an ssp where the bx is good enough, your beeping tool need never beep.
- (and you can have special procedures for when you do exit an ssp...)

Atlases of subspace pairs

Collection of ssps that covers both model spaces is called an atlas.

NB this means every $m \in M$ occurs in some pair (m, n) in some ssp

does not mean every pair (m, n) is in some ssp.

Example

Classic trouble-maker: `if`.

If class C is stereotyped persistent, then its methods must be matched by appropriate tests. Otherwise, C is not relevant to consistency.

Consider atlas of subspace pairs $\{(M_P, N_P) : P \subseteq \text{Classes}\}$

(exactly these classes stereotyped; at least those test classes exist)

Bx behaviour on an ssp is relatively straightforward and can have good properties.

What to do otherwise is more dangerous...

Tool beeps if set of persistent classes in the UML model has changed, or set of test classes has got too small.

Questions, comments?

It only remains for me to say...

Thank you Jeremy!