Model-Driven Engineering and the Impact of a Change

Alexander Egyed

Johannes Kepler University (JKU), Linz, Austria

http://www.sea.jku.at

Who am I?

JOHANNES KEPLER UNIVERSITY LINZ | JKU

Current Affiliations:

- Professor at **Johannes Kepler University**, 2008
- Head of **Institute for Systems Engineering and Automation** (~14 Staff Members)
- Research Fellow at **IBM**, 2010-12

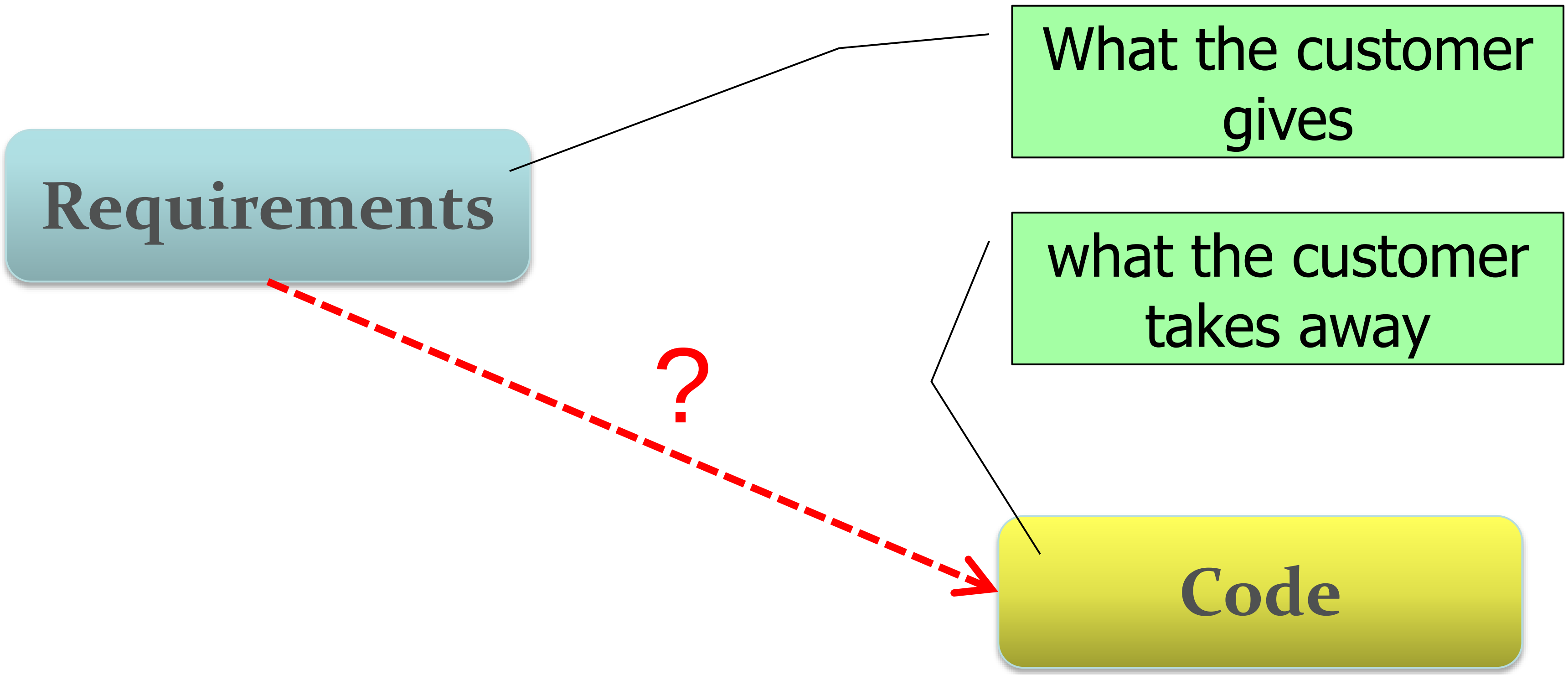
Doctorate Degree:

- University of Southern California, USA 2000 (Dr. Boehm)

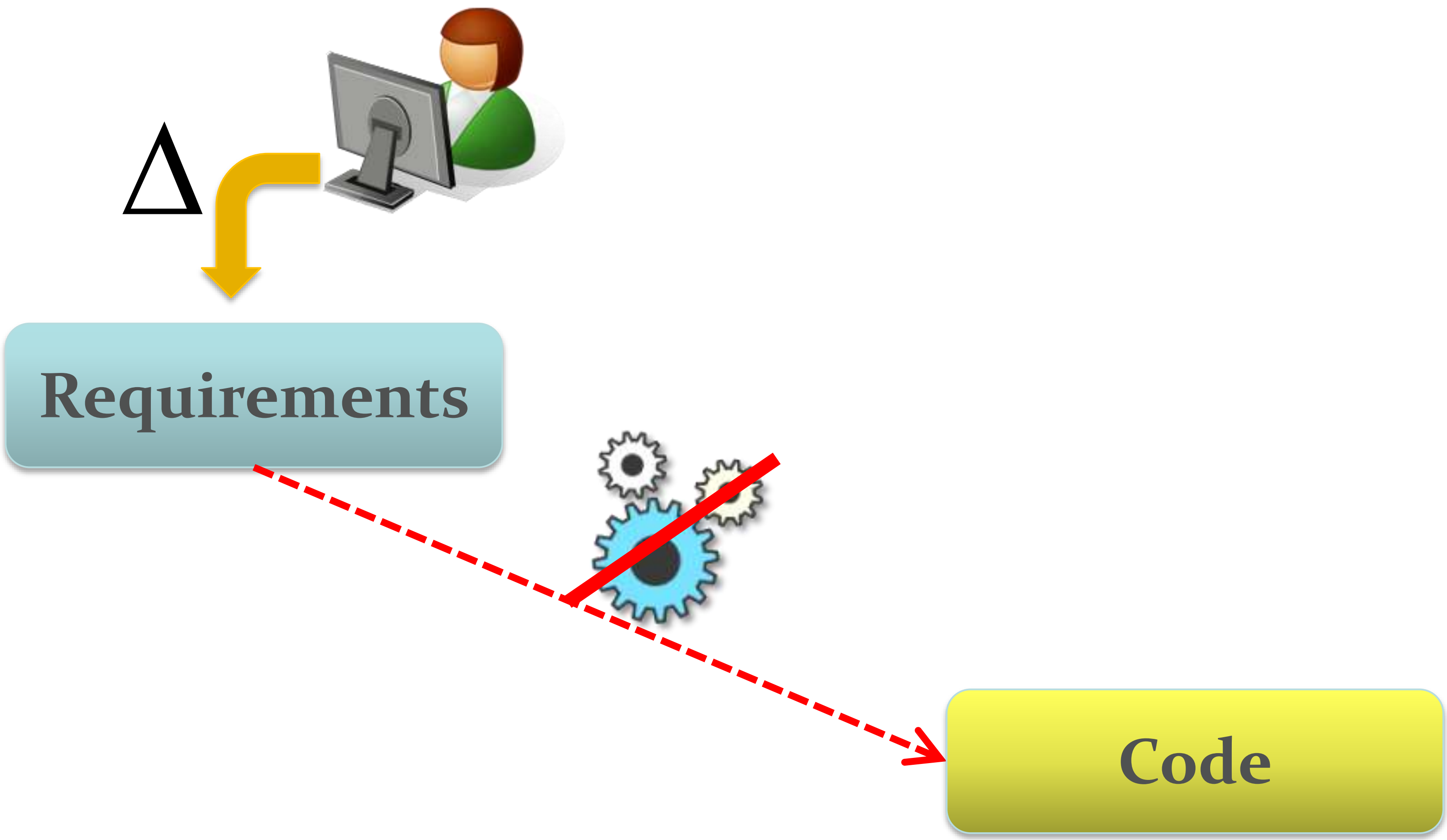
Past Affiliations:

- Research Fellow at University College London, UK 2007
- Research Scientist at Teknowledge Corporation, USA 2000

What the Customer cares about...



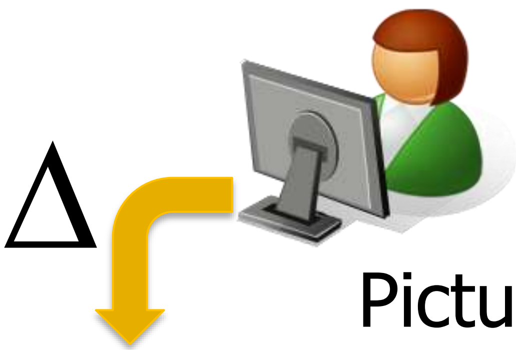
Focus on Change



Impact of a Change

- Changes can happen anywhere / anytime
  - Requirements change, infrastructure change, law change...
- A change is a „small“ thing
- Inability to change a software system is one of the foremost software engineering challenges

Models Complicate this Relationship



THE GOOD?

Analyses / proofs

Picture says more than a 1000 words

Important design decisions

It is good engineering

…

Requirements → **Design Model** → **Code**

THE BAD?

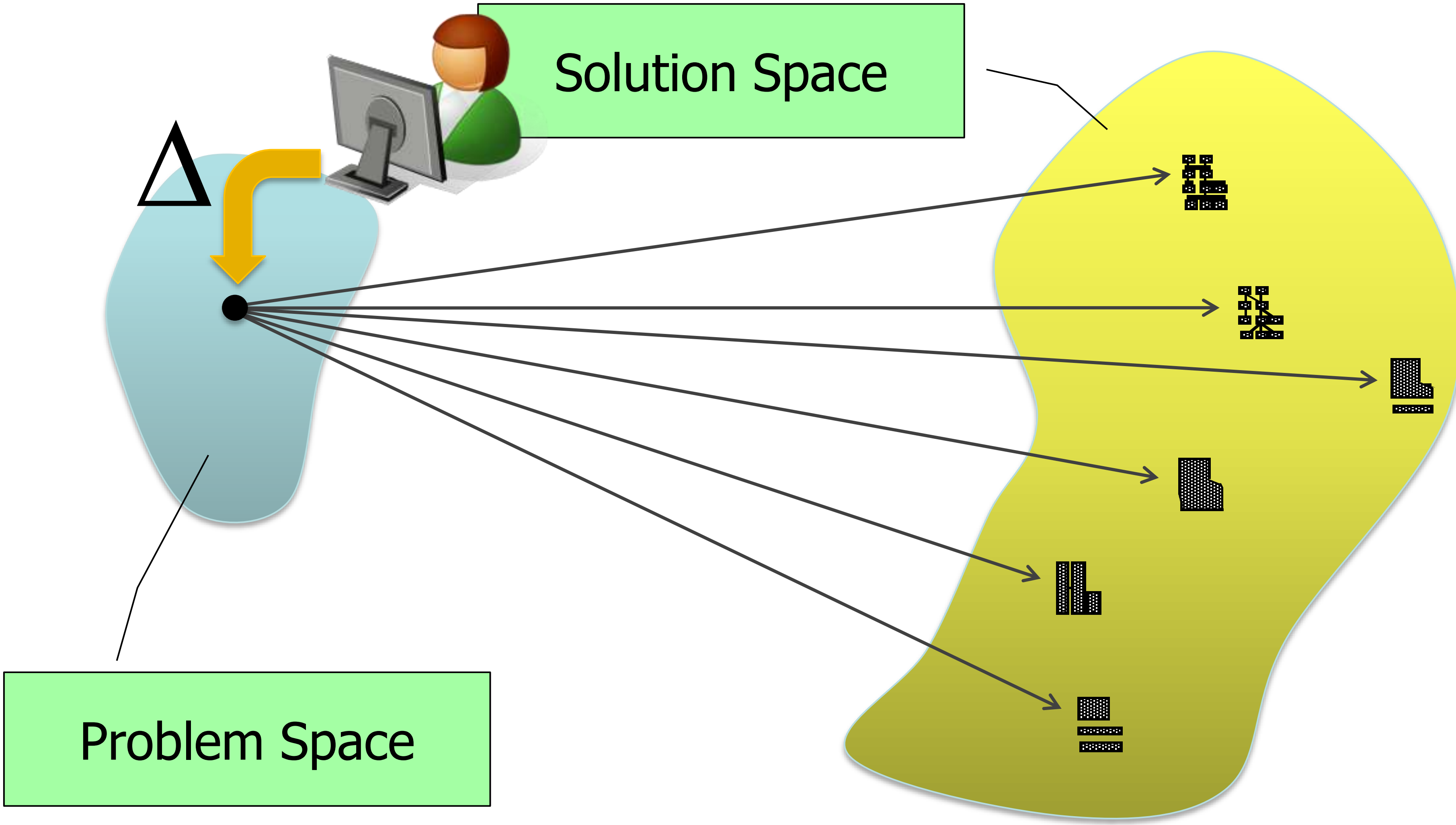
Model Dilemma

JOHANNES KEPLER UNIVERSITY LINZ | JKU

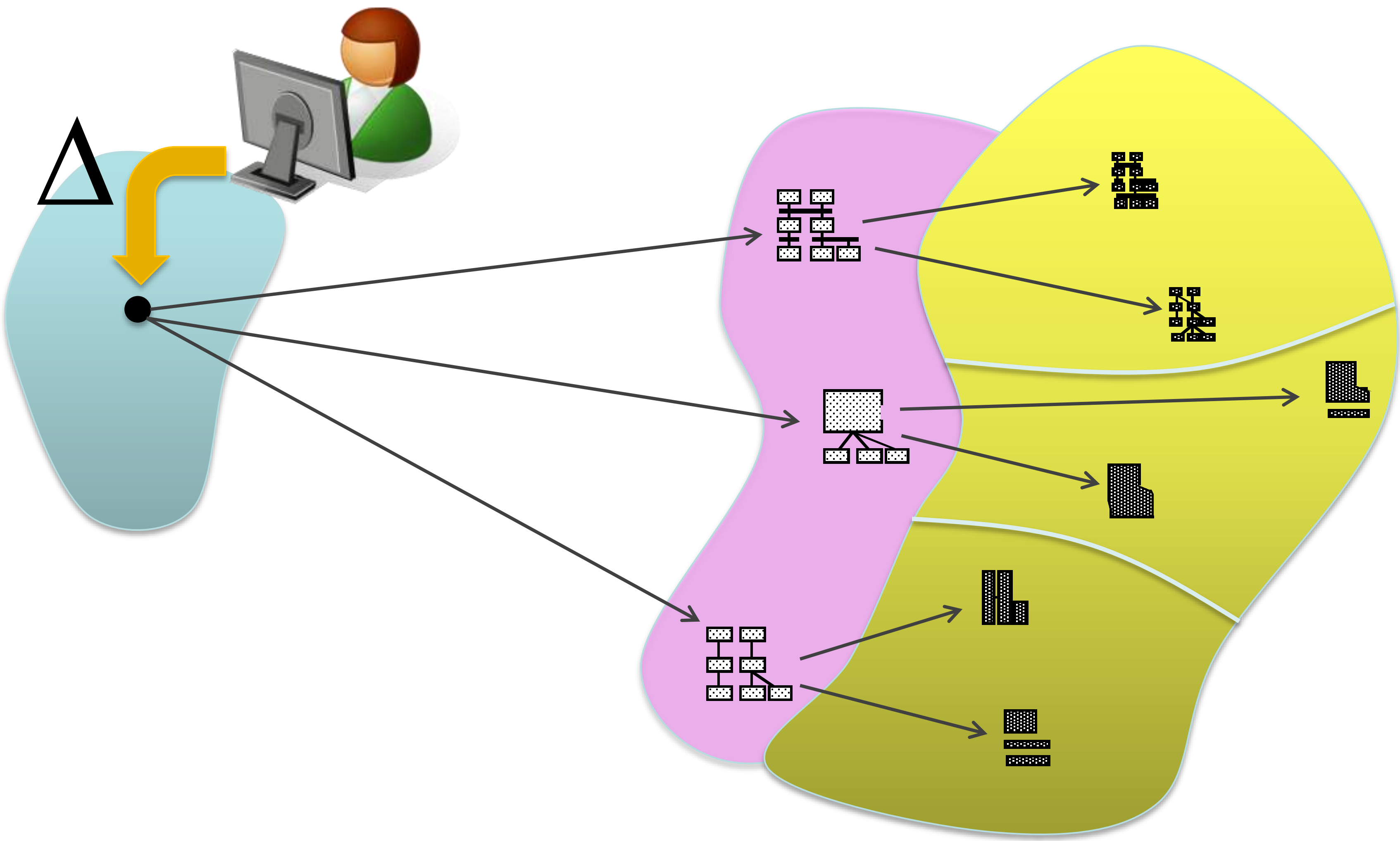
Nobody wants to MAINTAIN them

- Maintaining models is a burden
- Models were not made for change propagation
  - Just like code, requirements, …

Many Solutions to a Given Problem…

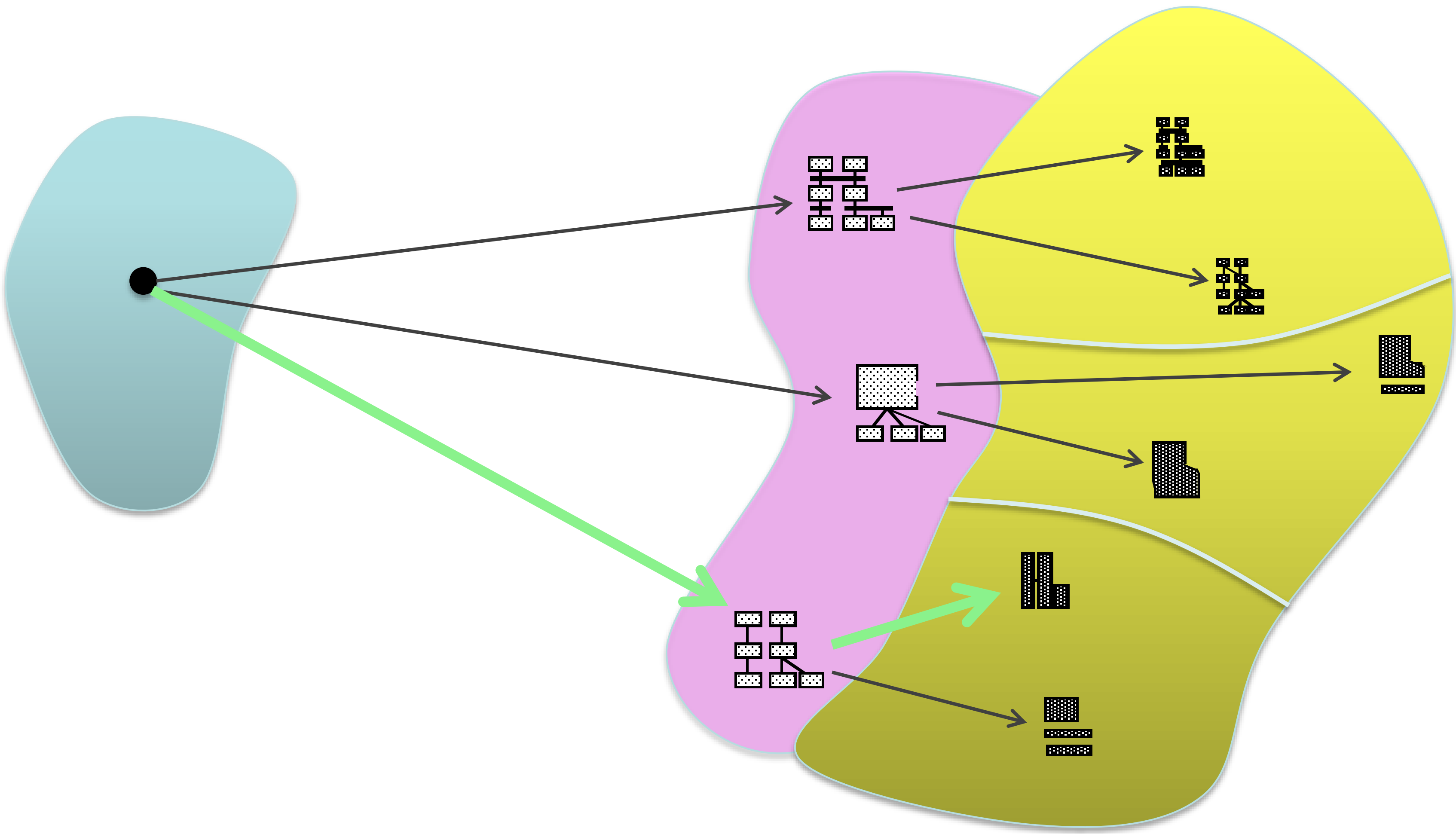


Design Model Restricts the Solution



Slide adapted from Nenad Medvidovic

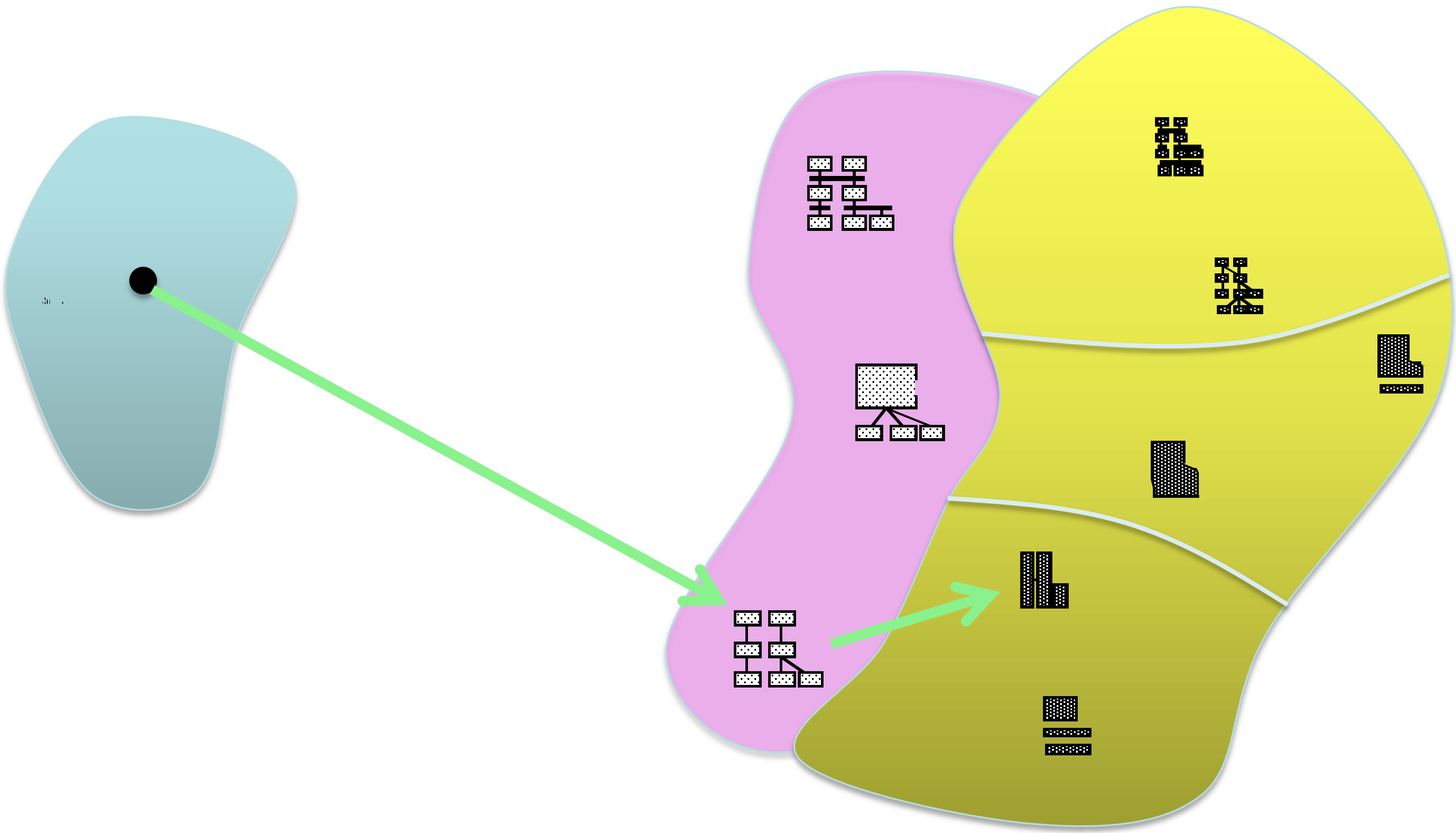
Design Model Helps you Choose a Solution



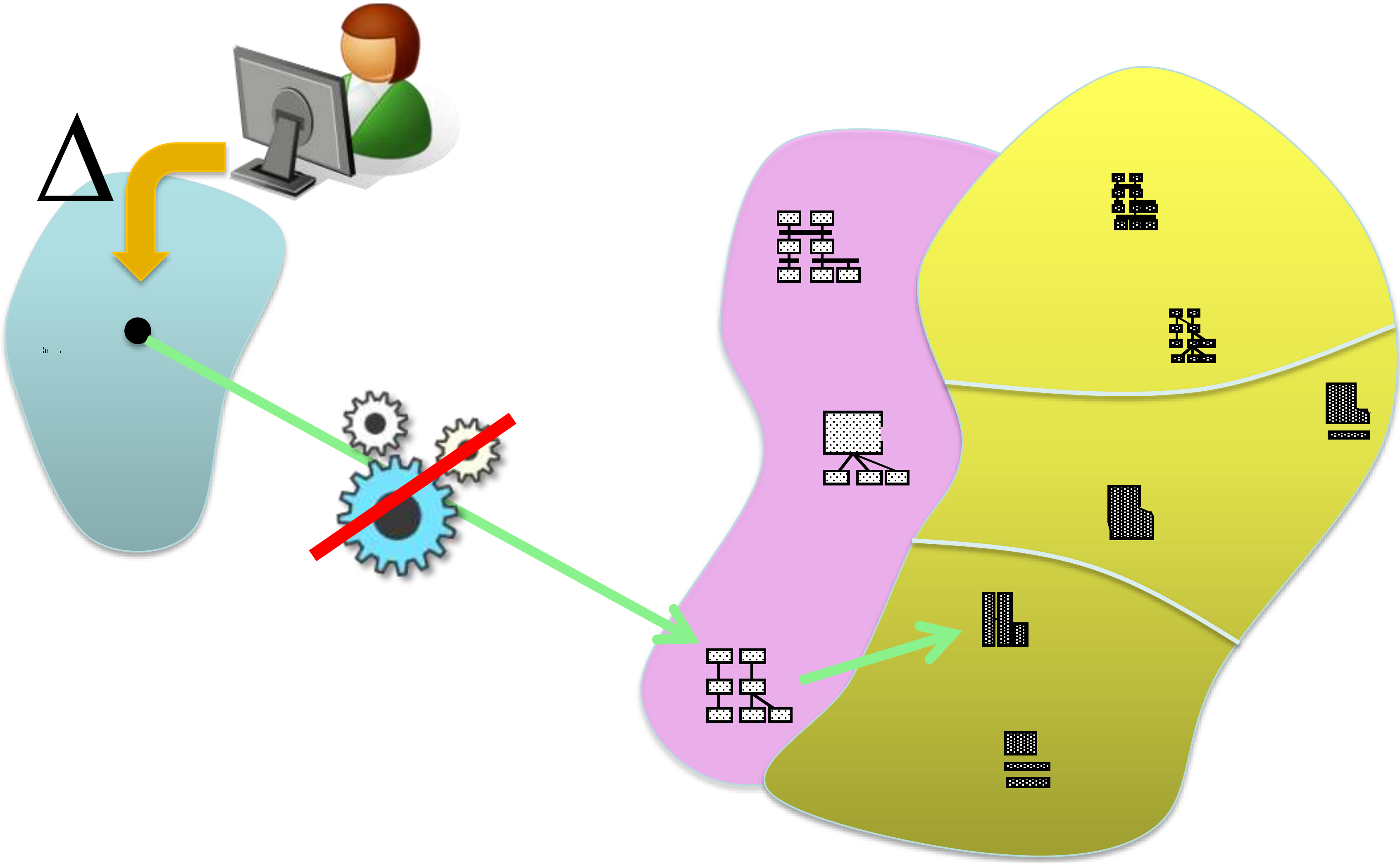
Problem

- Models as a “Bridge” for Change Propagation
- But creating AND MAINTAINING THEM is still a burden, or is it?

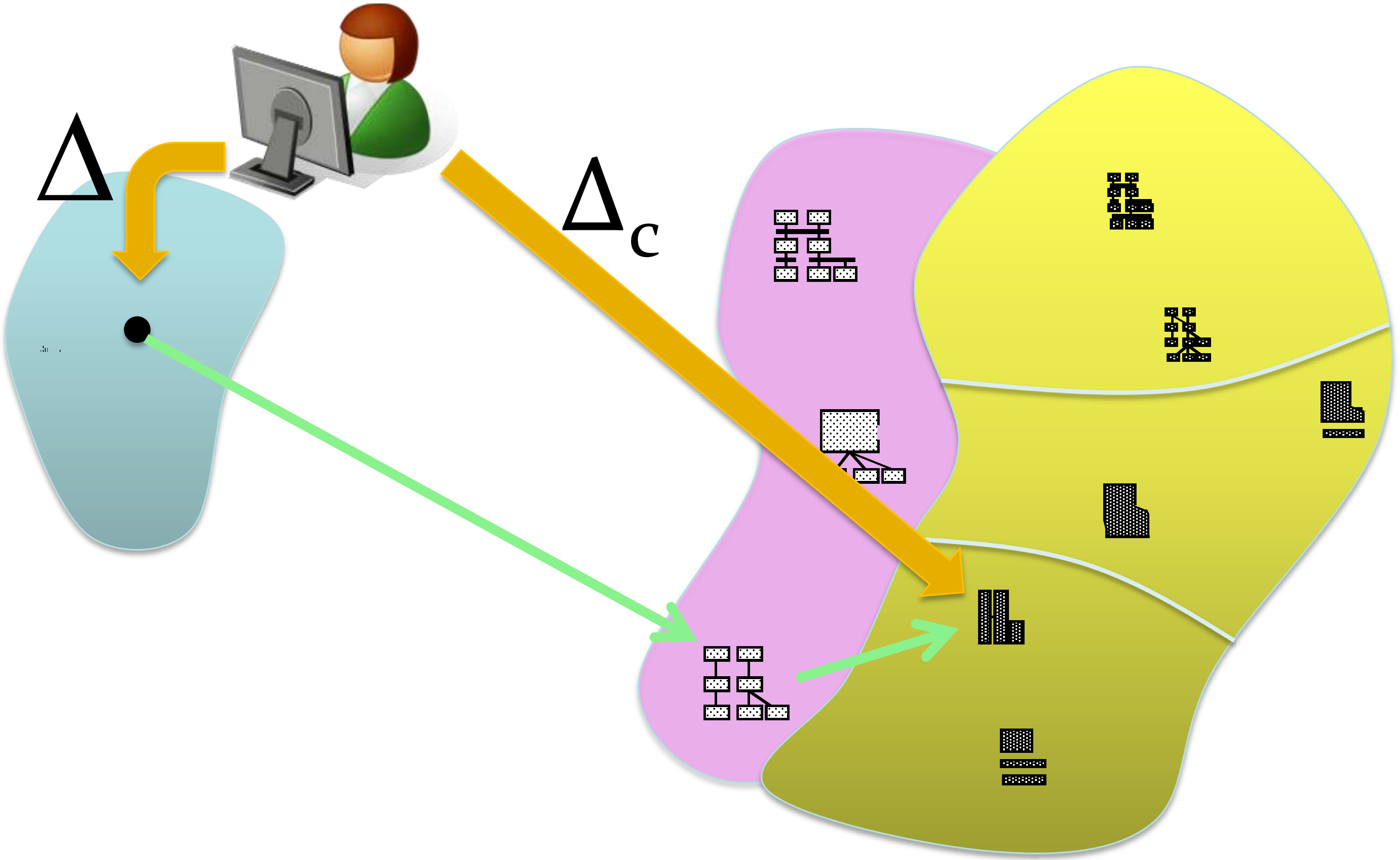
Maintaining the Model



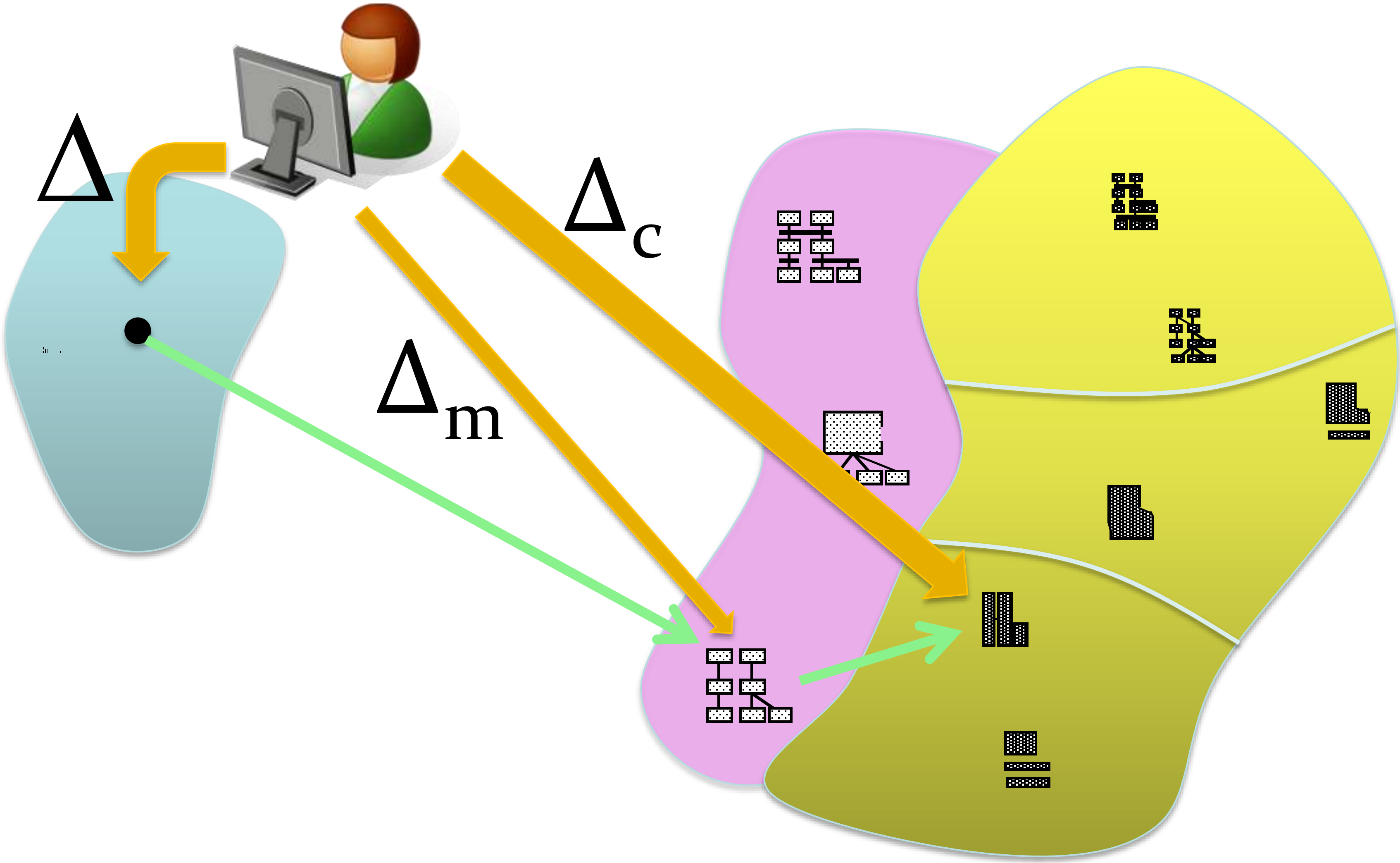
Maintaining the Model



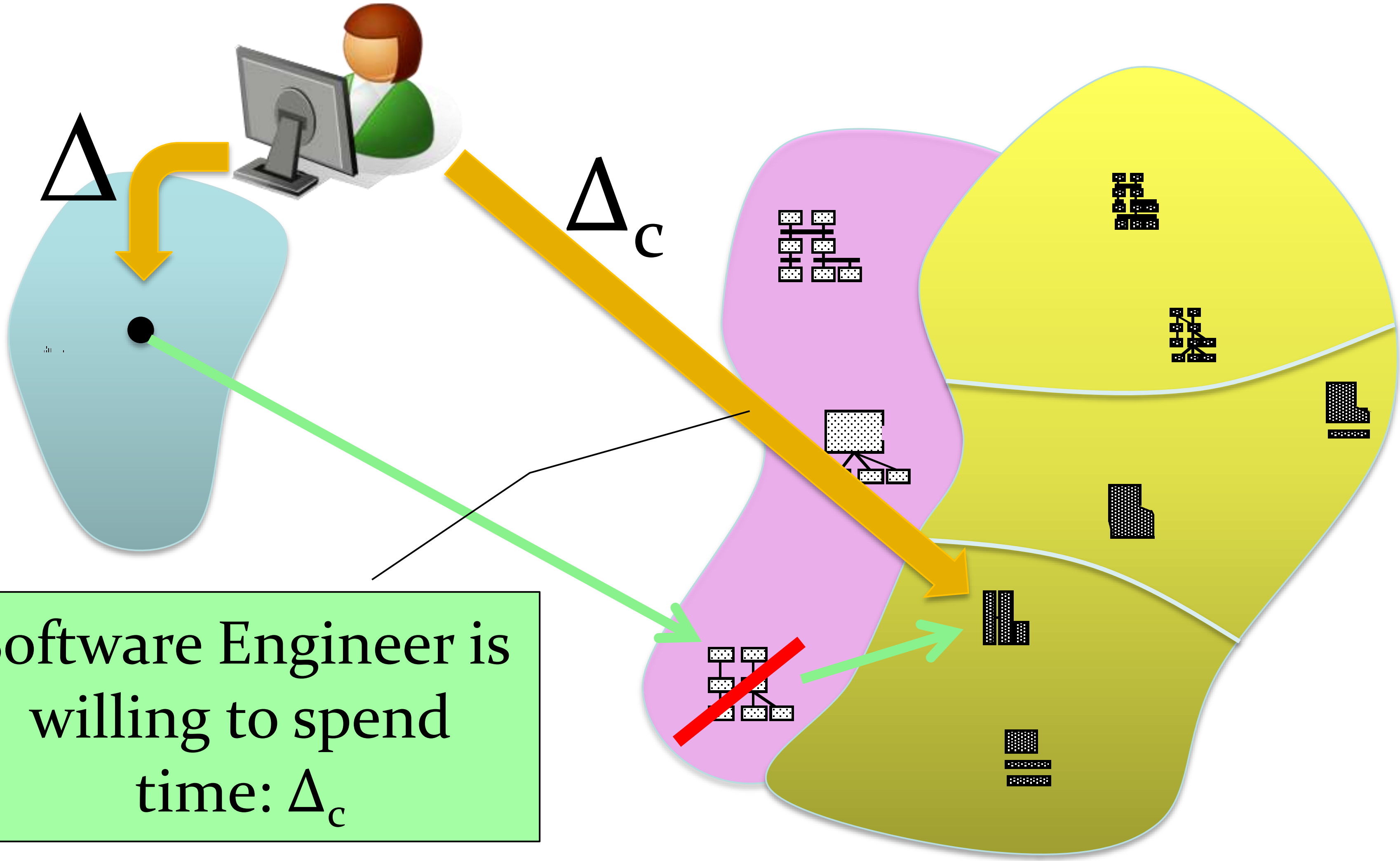
Maintaining the Model



Maintaining the Model

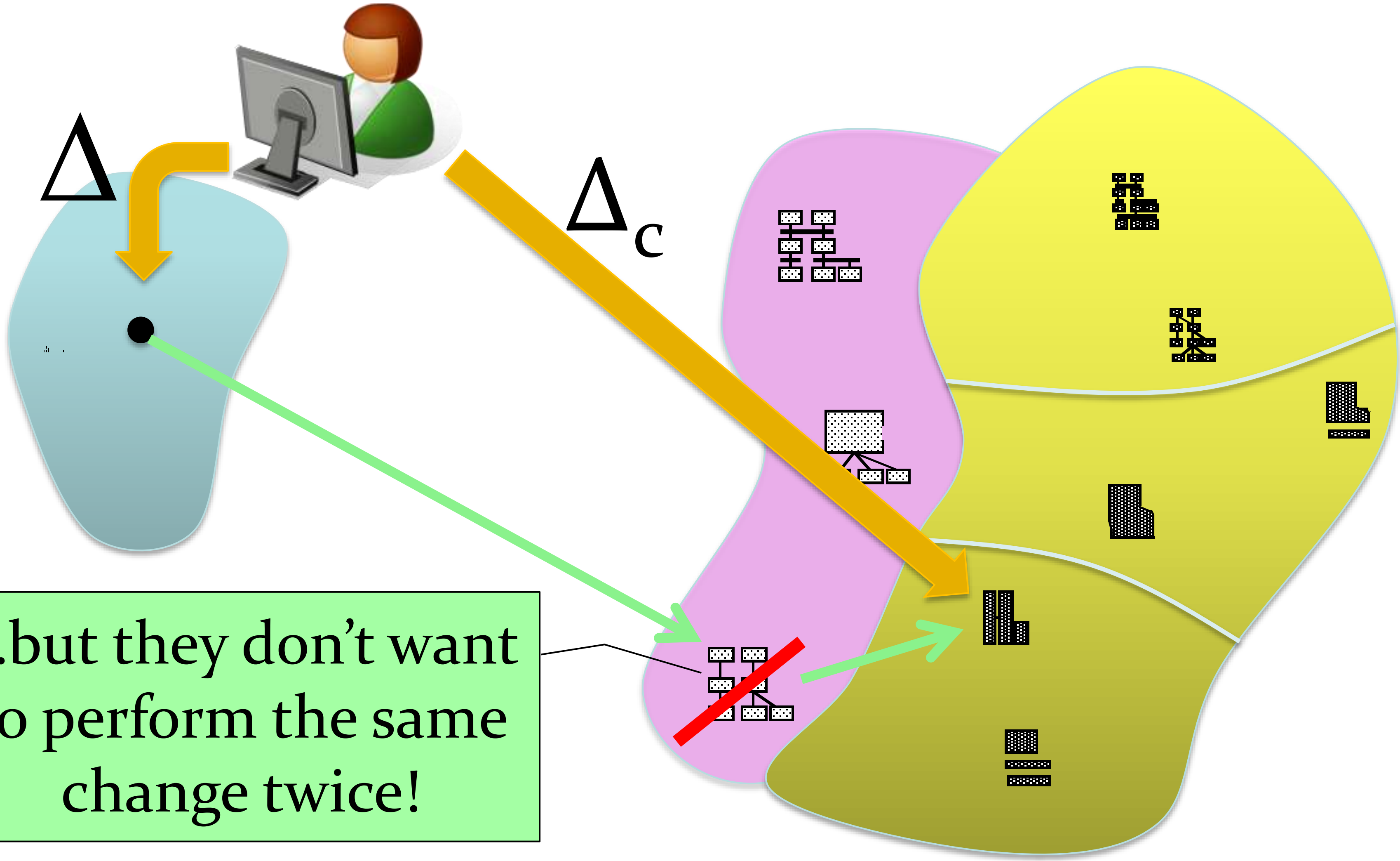


Maintaining the Model



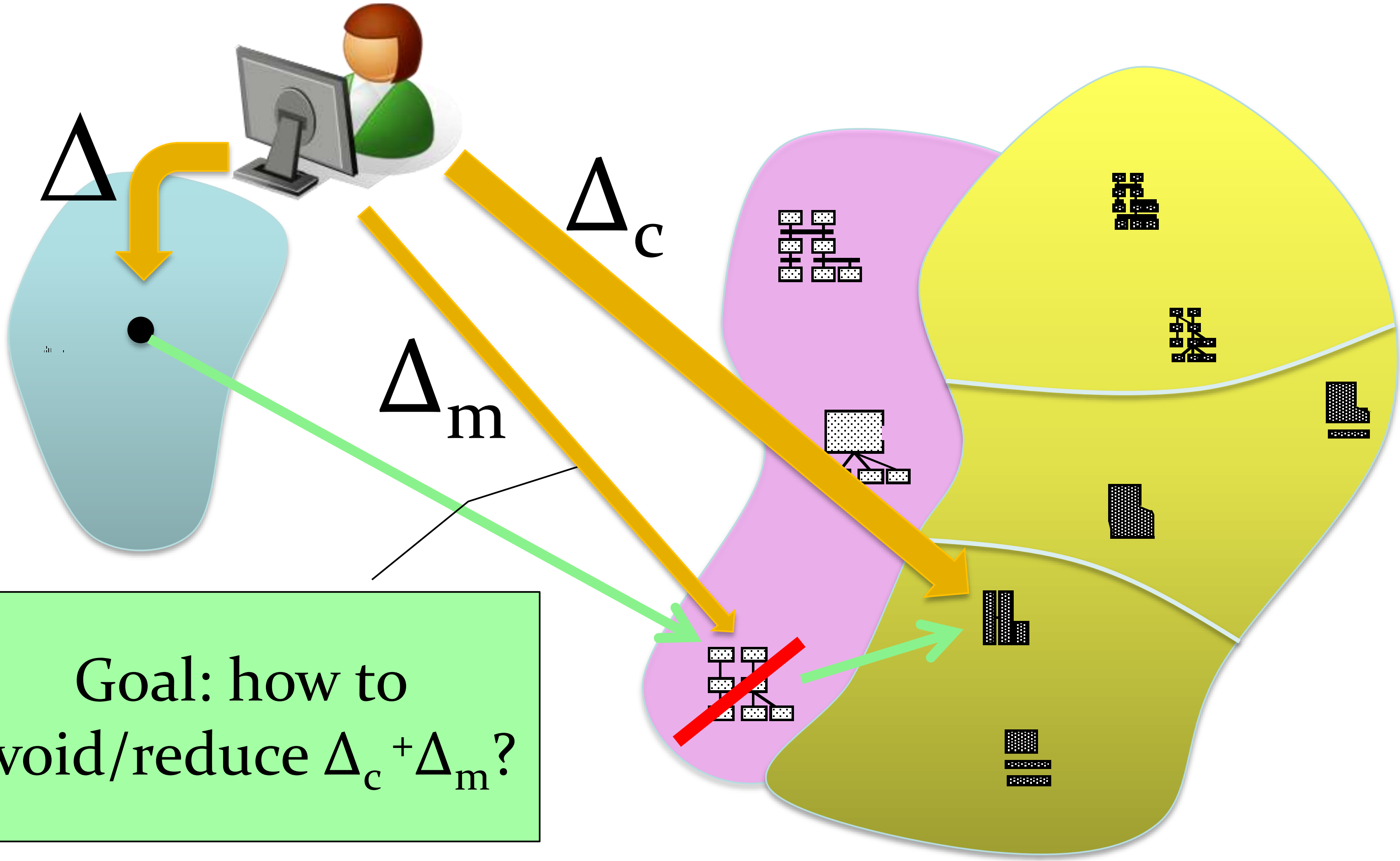
Software Engineer is willing to spend time: Δ_c

Maintaining the Model



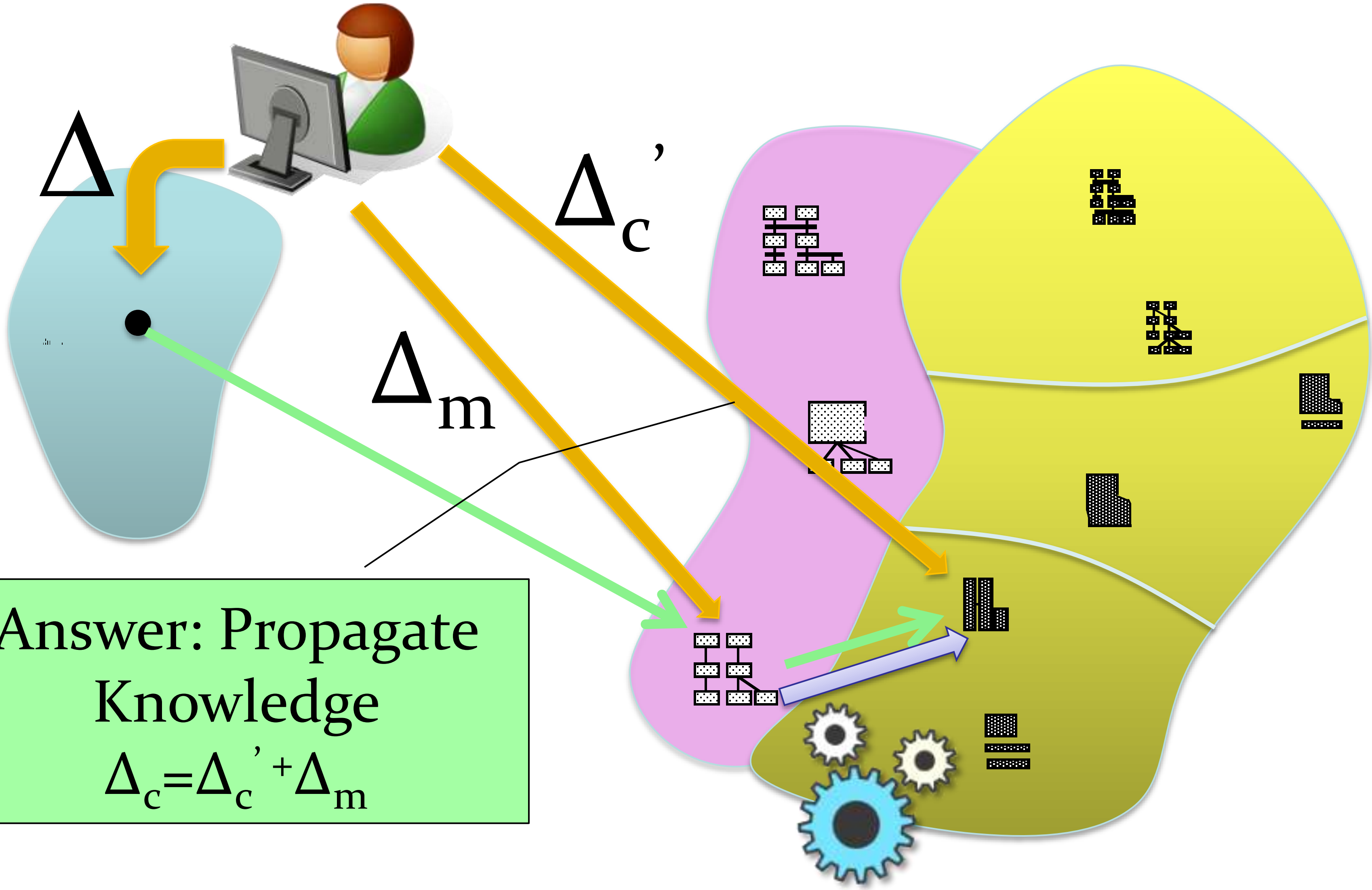
...but they don't want to perform the same change twice!

Maintaining the Model



Goal: how to avoid/reduce $\Delta_c + \Delta_m$?

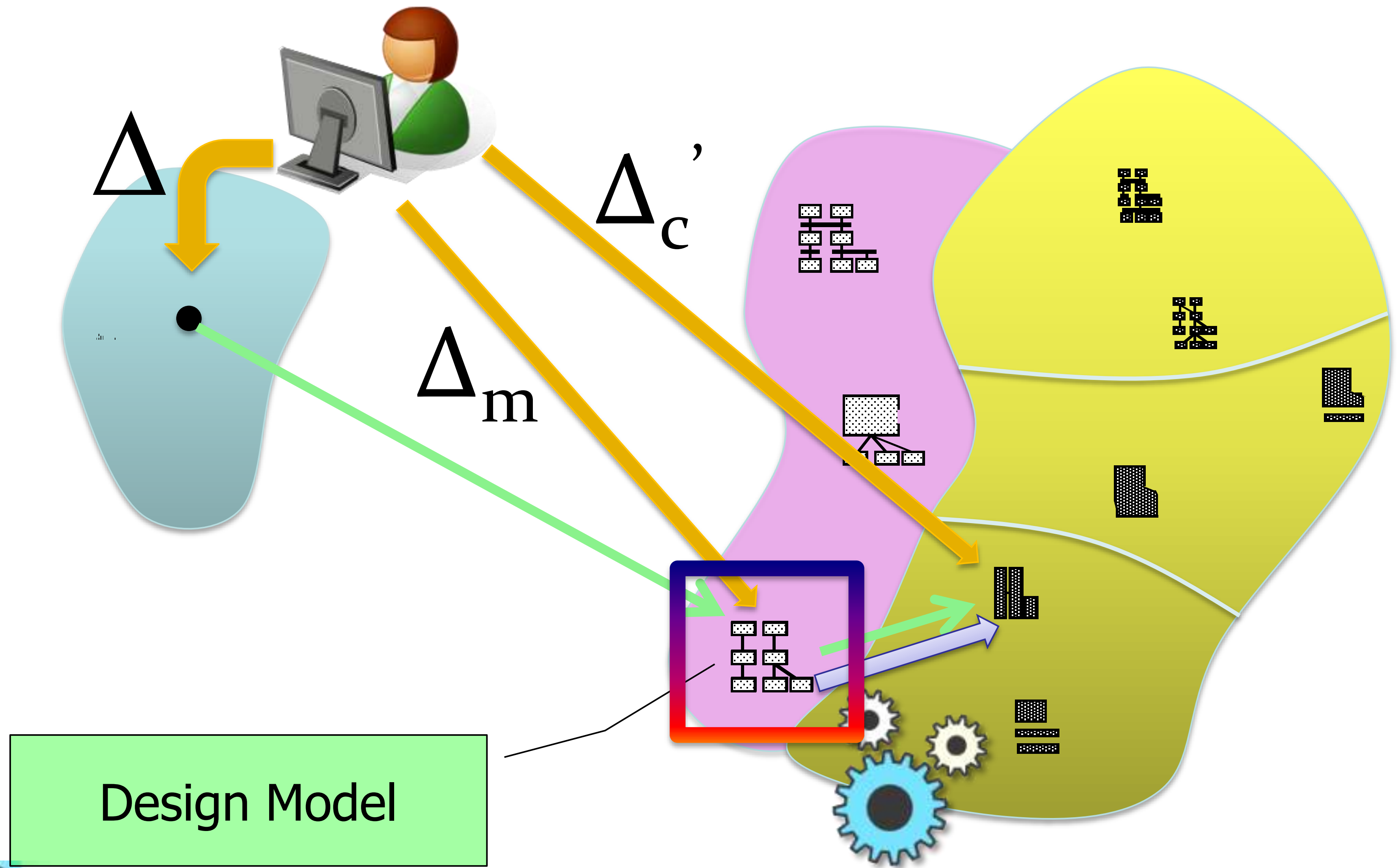
Maintaining the Model



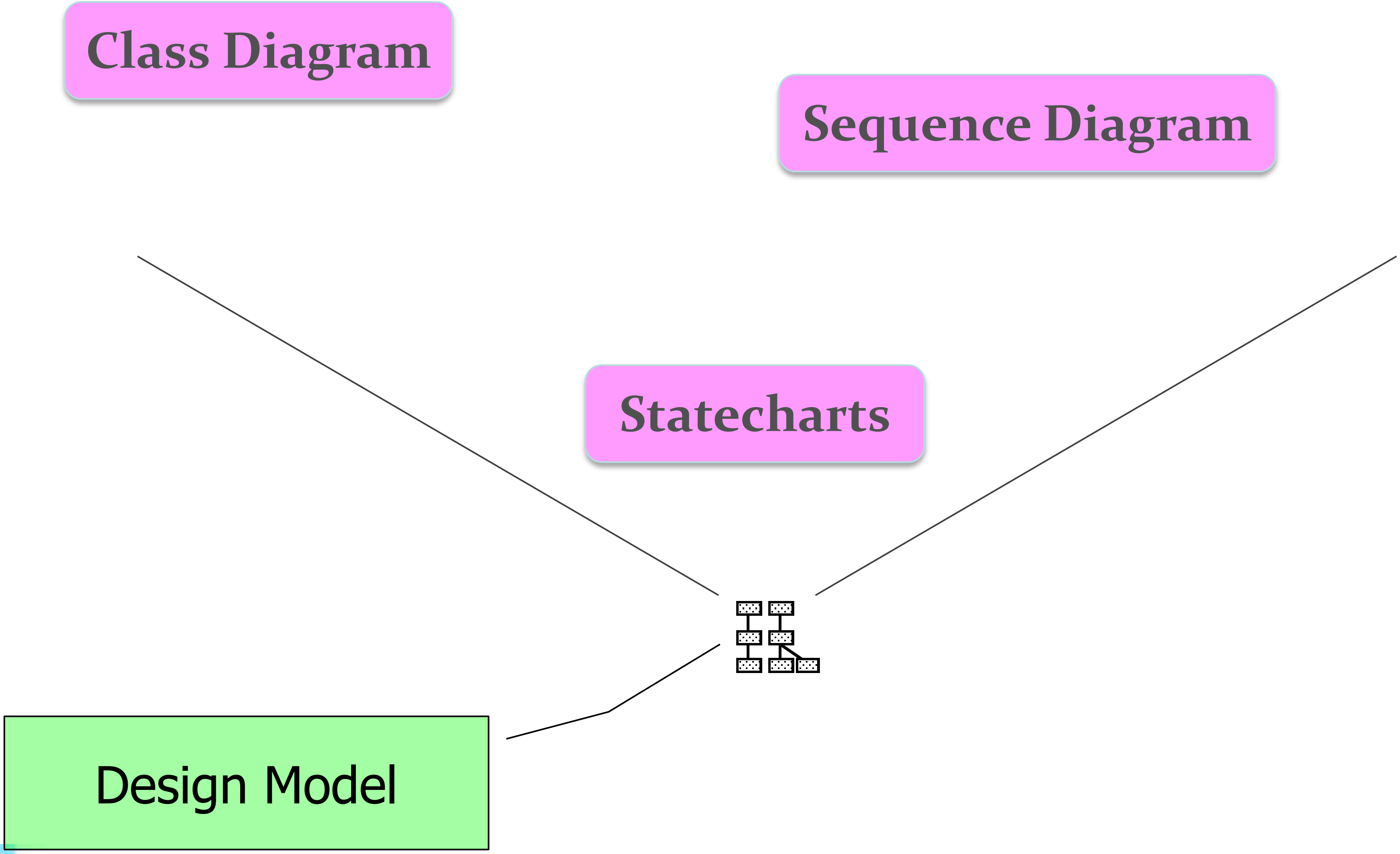
Answer: Propagate Knowledge

$\Delta_c = \Delta_c' + \Delta_m$

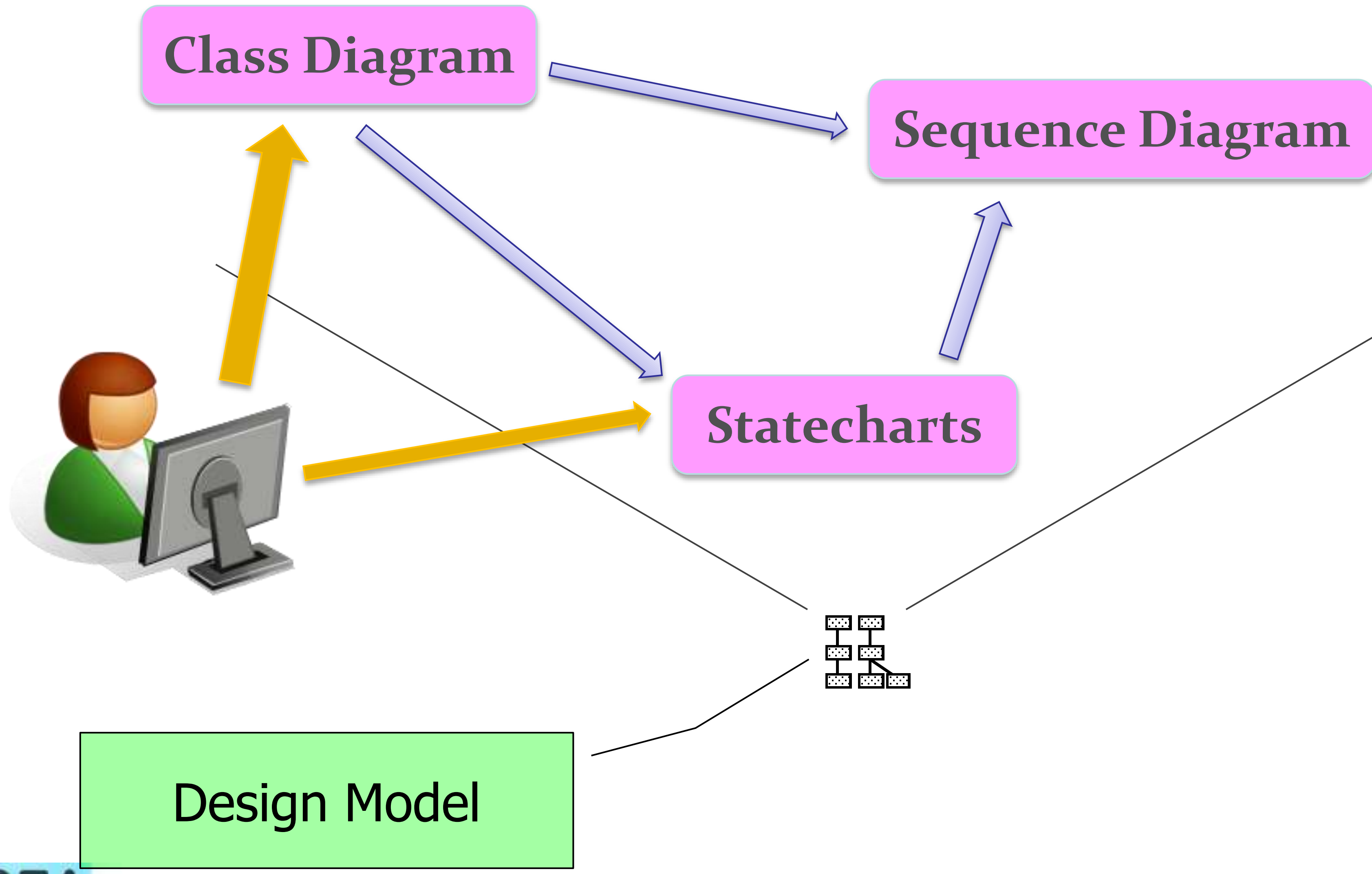
There are Many Models…



There are Many Models…



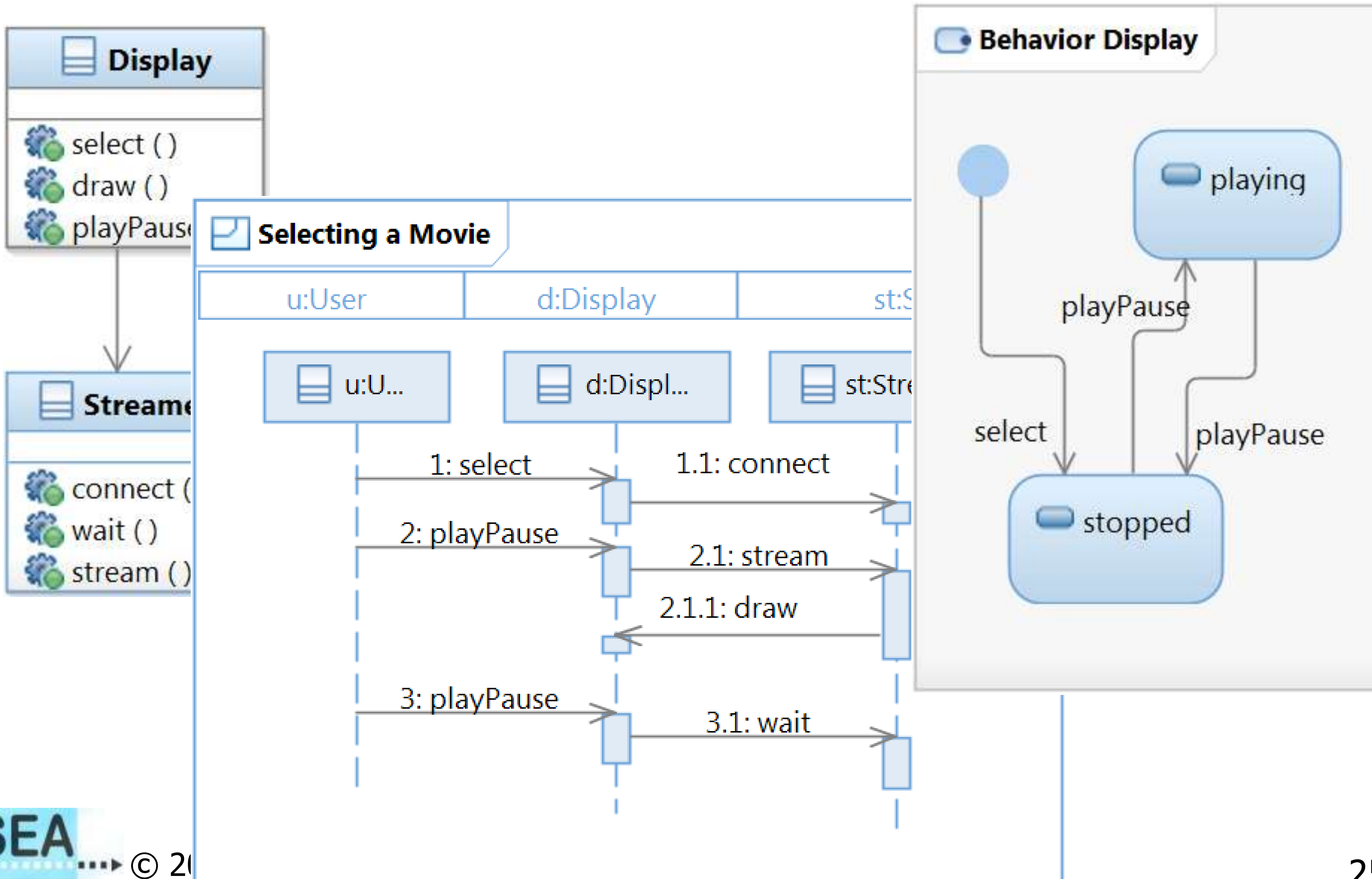
...to Propagate Changes to



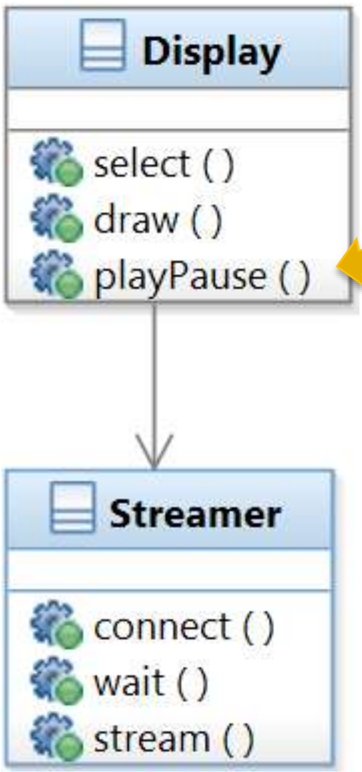
A Motivating Illustration for Change Propagation

(propagating changes, not models)

Modeling Languages are Diverse



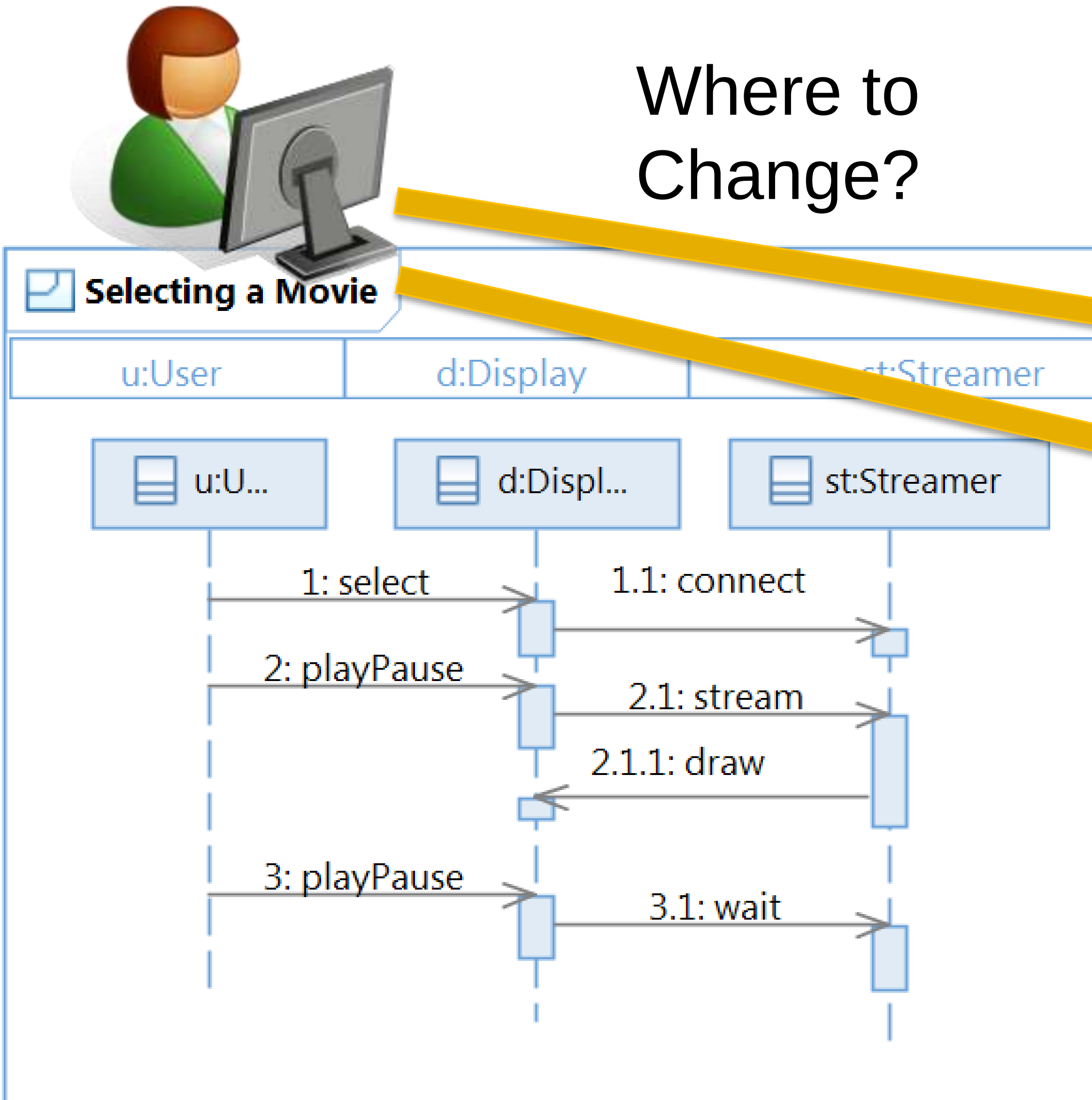
A Change

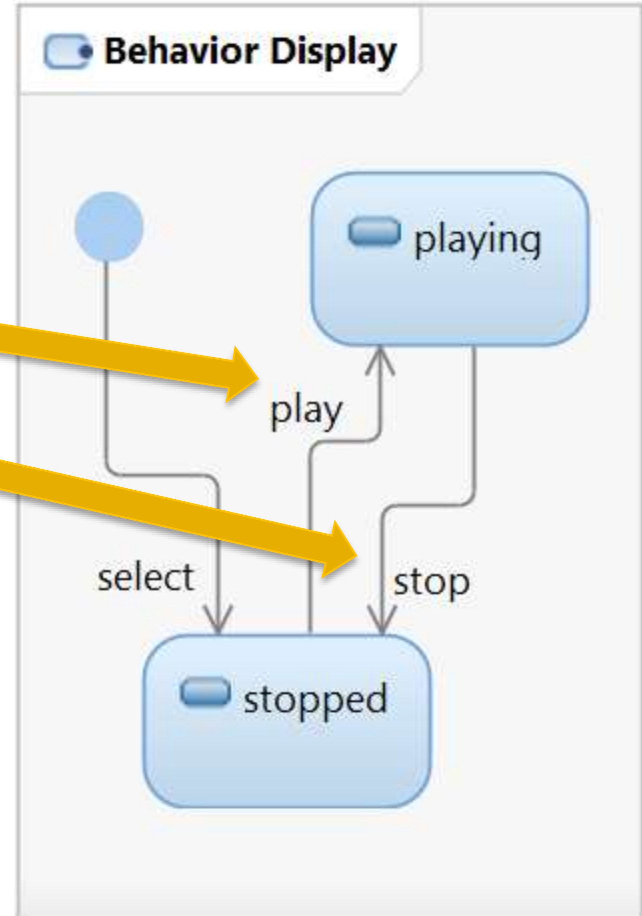


Split
“playPause()”
into “play()”
and “stop()”

Change Propagates

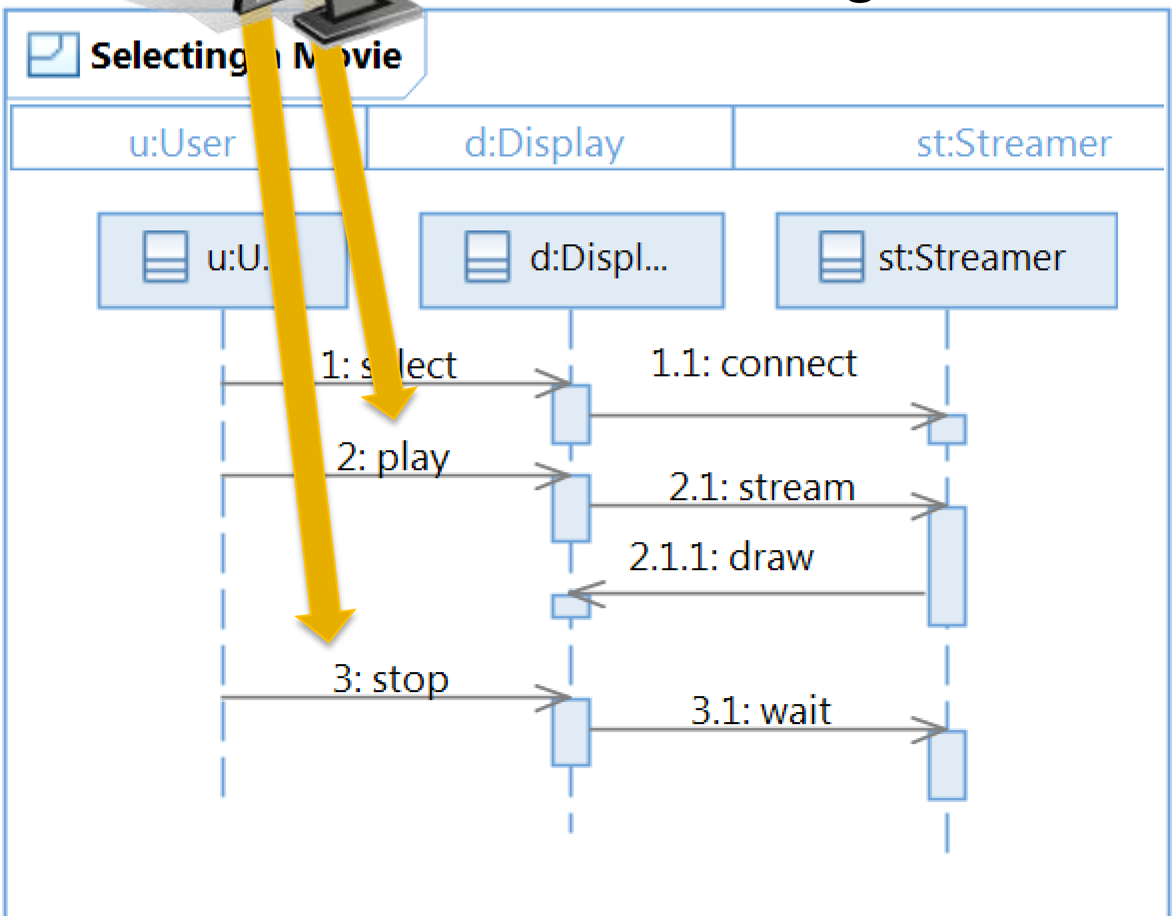
Where to Change?

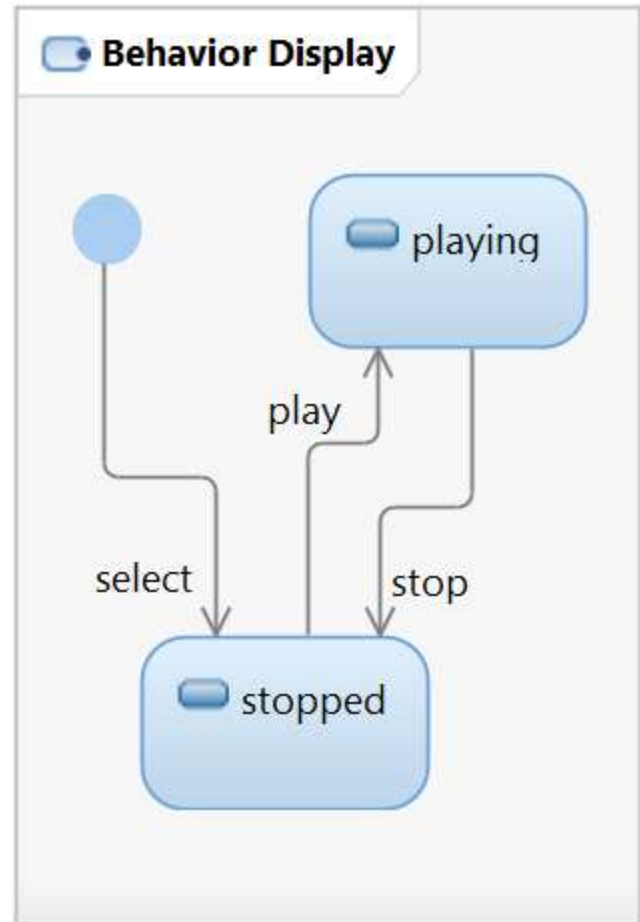




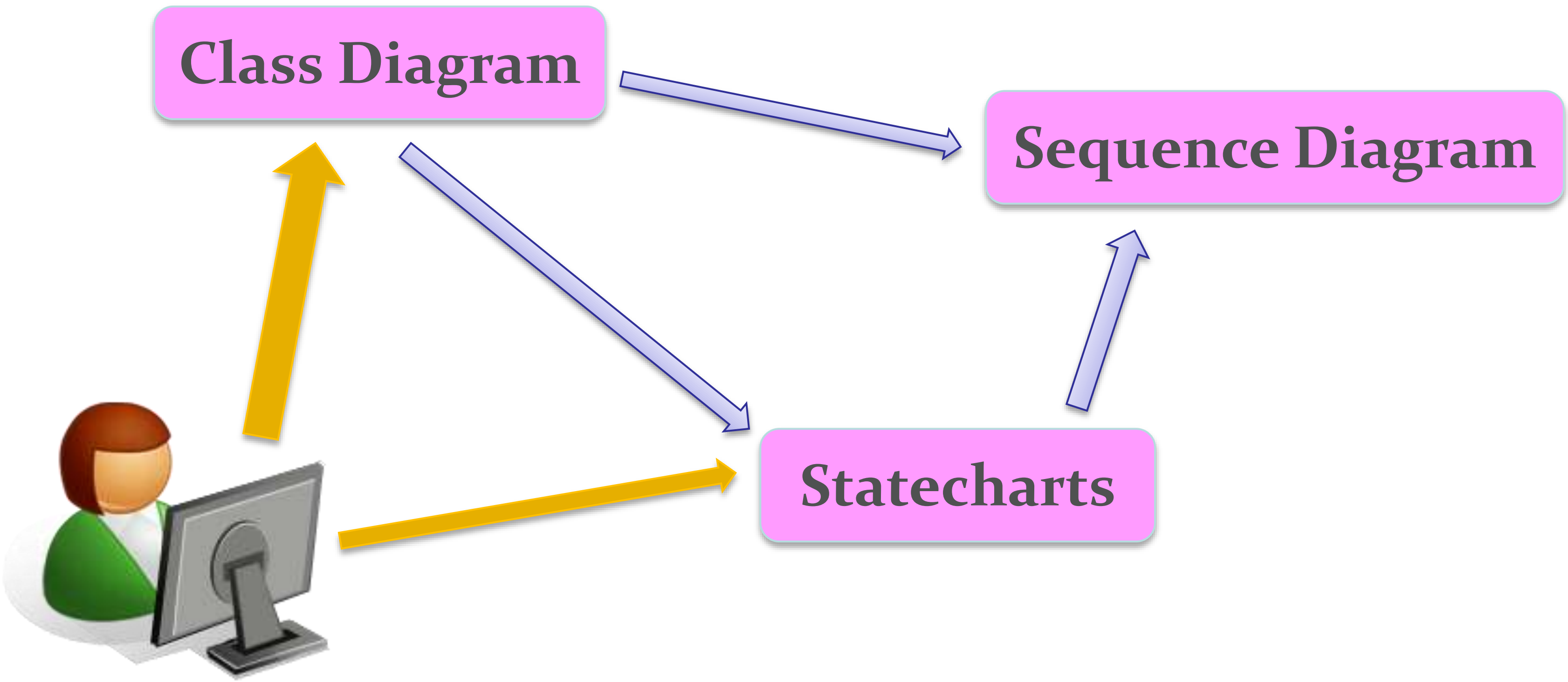
Change Propagates

How to Change?





Change Propagation is…



- Where to Change (Locations)
- How to Change (Values)

Constraint-Driven Change Propagation

- This is not about designing automatically
  - The software engineer designs
  - The automation only propagates their logical conclusions
    - More often constraints rather than model elements
- Designing is "fully manual"

Where to Change

Tool

Rename playPause() operation to play(). Show Design Rules.

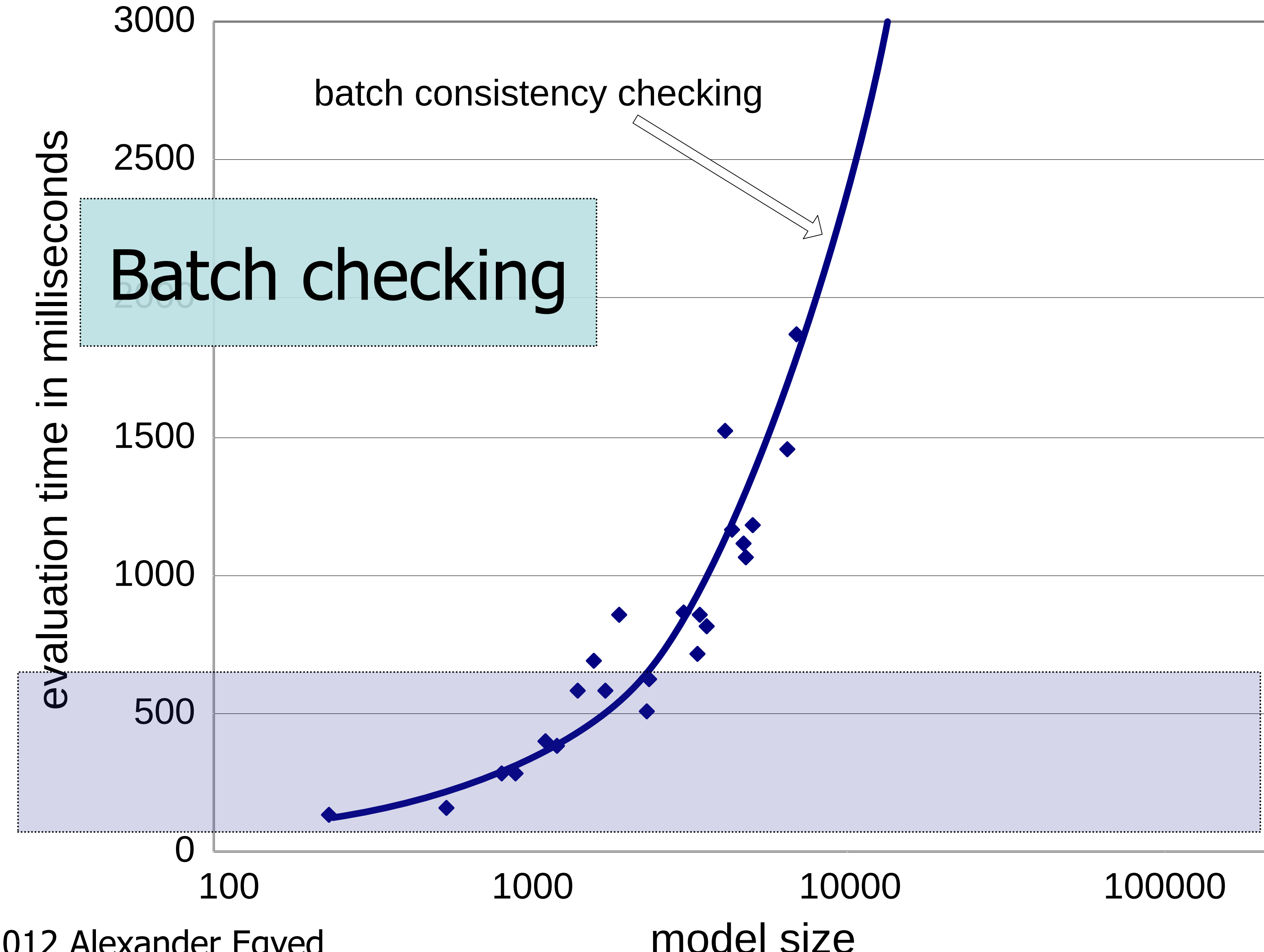
Detect inconsistencies instantly (evaluation tree)

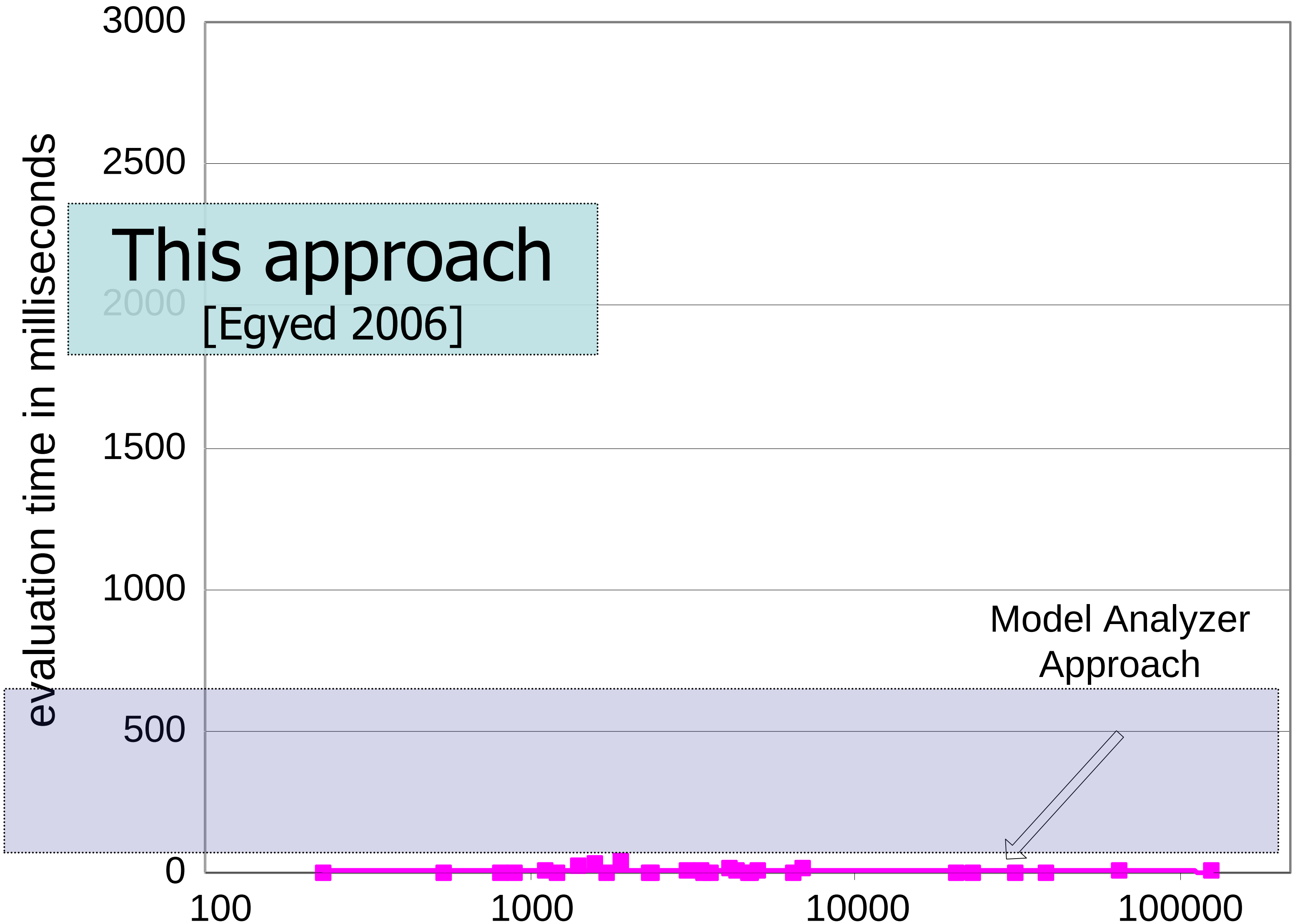
Two Advances

JOHANNES KEPLER UNIVERSITY LINZ | JKU

1) We treat every evaluation of a consistency rule as a first class citizen – by maintaining change impact scopes for them individually and triggering individual re-evaluations
2) We use model profiling to observe the "behavior" of consistency rules during their evaluation to automatically compute change impact scopes

Batch checking




This approach
[Egyed 2006]
Model Analyzer
Approach
evaluation time in milliseconds
3000
2500
2000
1500
1000
500
0
100
1000
10000
100000
model size

Implications

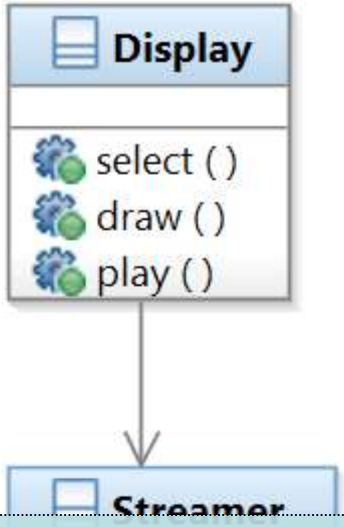
- We can quickly evaluate model changes
- And we can identify which model elements resolve inconsistencies (where to change)

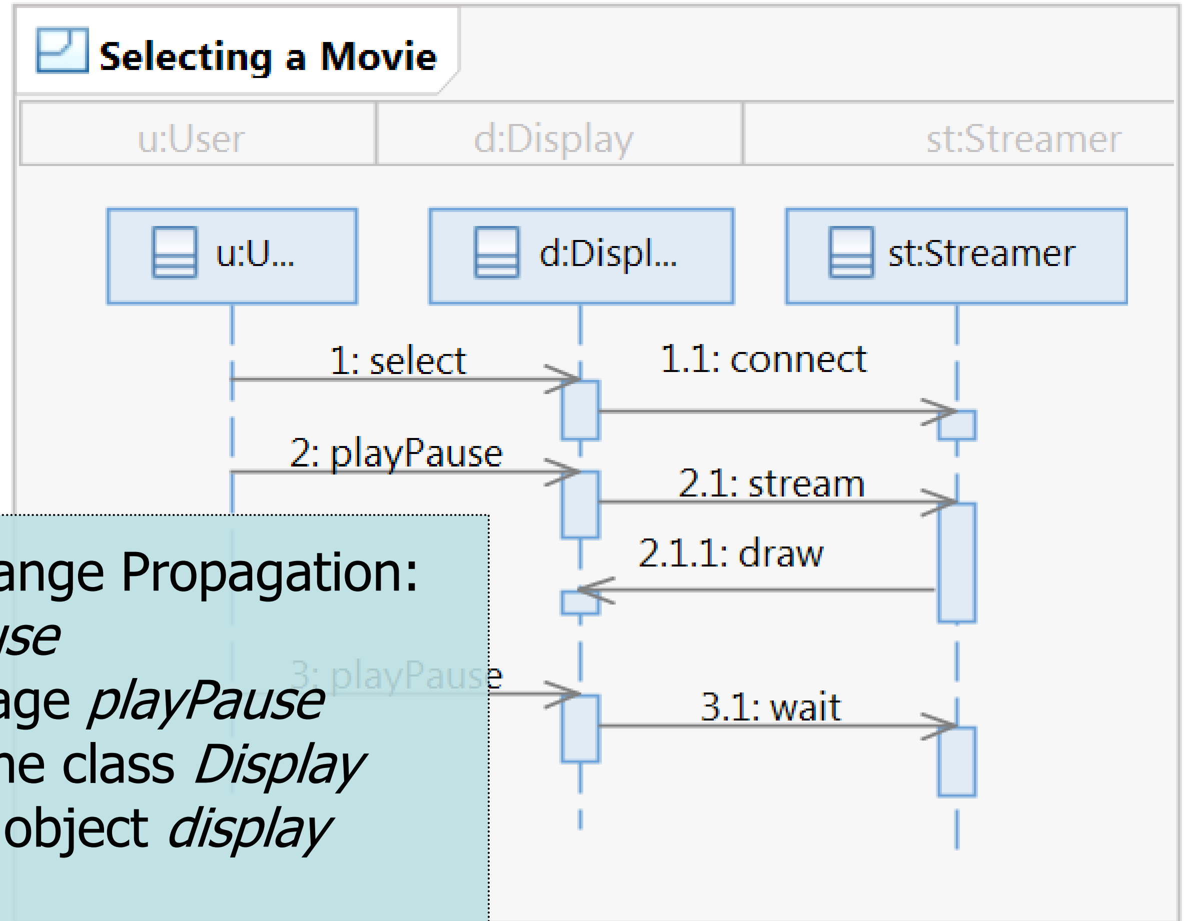
How to Change

Tool

Enumerate repair alternatives affected by renamed operation

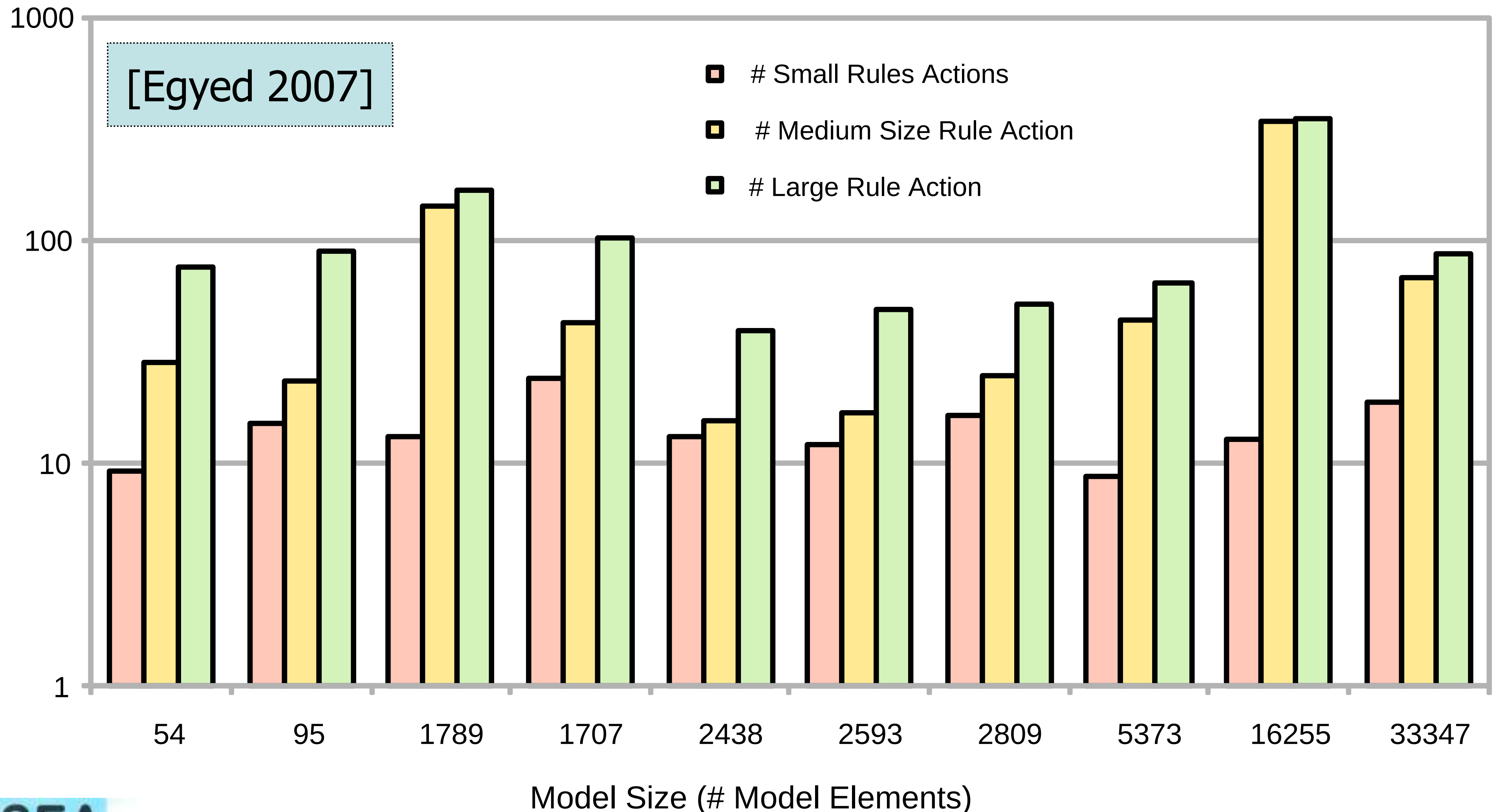
“playpause” to “play”





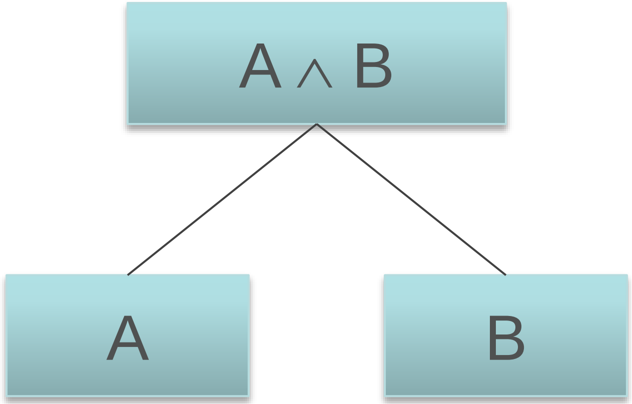
Alternative Locations for Change Propagation:
1) rename message *playPause*
2) Change receiver of message *playPause*
3) add a new operation to the class *Display*
4) change the ownership of object *display*
6) rename operation *select*
7) rename operation *play*
8) rename operation *draw*
9) delete message *playPause*

Quite good but not Perfect



To propagate changes, you must understand the design rules

Not every element needs fixing

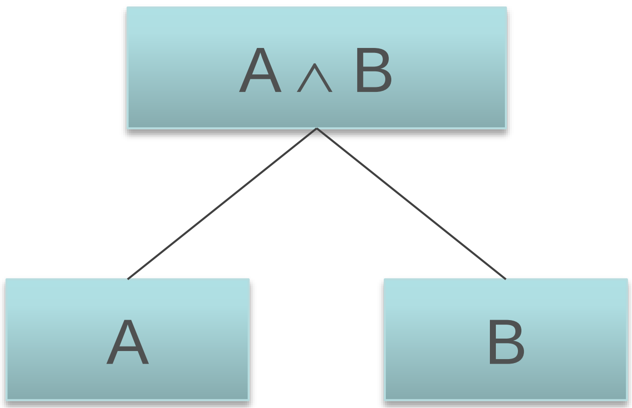


Fixing:

if A ∧ B = false then either A needs fixing, B needs fixing, or both A and B need fixing.

[Nentwich, Emmerich, and Finkelstein 2003]

Not every element needs fixing

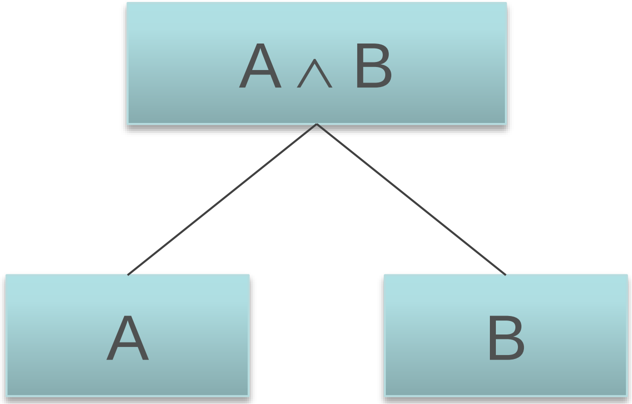


Fixing:

If A is true then we need not fix A if $A \wedge B$ = false

[Reder-Egyed 2012]

Not every element needs fixing



Fixing:

If A is true then we need not fix A if $A \wedge B$ = false

[Reder-Egyed 2012]

Fixing Actions for $A \wedge B$

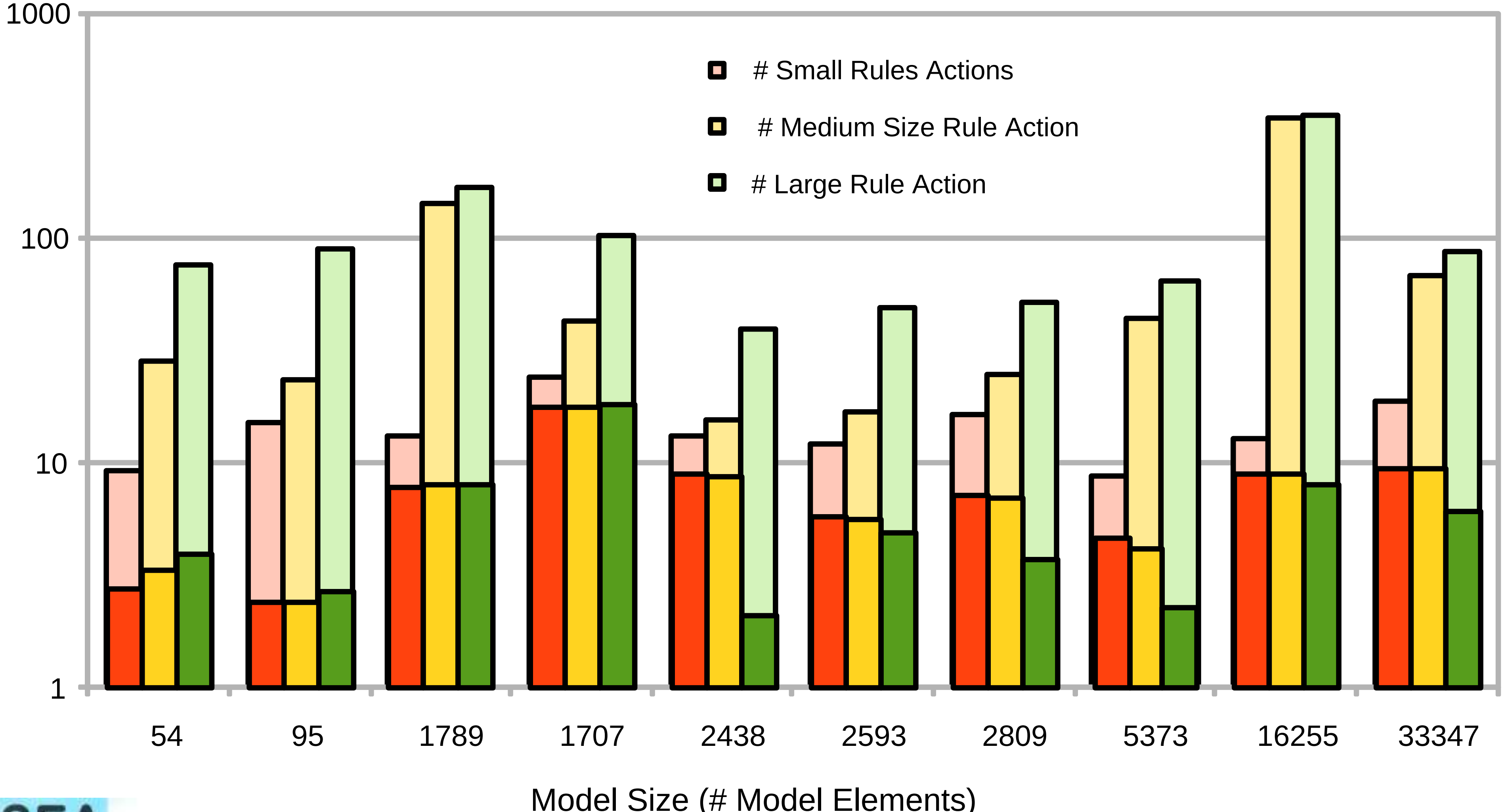
Required Result	Evaluated Result	Fixing Action
True	A=true and B=false	Fix B=true
True	A=false and B=true	Fix A=true
True	A=false and B=false	Fix ⊗[A=true, B=true]
False	A=true and B=true	Fix ● [A=false, B=false]

Required Result for $A \wedge B$ = false if $\neg (A \wedge B)$

Repairs

o	α	R
$\neg$	$\{a\}$	$G(a,\ \neg r^e)$
$\wedge$	$\{a,\ b\}$	$R = \begin{cases} G(b\ r^e) & \text{if } r^e = t,\ r_a^v = t,\ r_b^v = f \\ G(a,\ r^e) & \text{if } r^e = t,\ r_a^v = f,\ r_b^v = t \\ G(a,\ r^e) \bullet G(b,\ r^e) & \text{if } r^e = t,\ r_a^v = f,\ r_b^v = f \\ G(a,\ r^e) + G(b,\ r^e) & \text{if } r^e = f,\ r_a^v = t,\ r_b^v = t \end{cases}$
$\vee$	$\{a,\ b\}$	$R = \begin{cases} G(a,\ r^e) + G(b,\ r^e) & \text{if } r^e = t,\ r_a^v = f,\ r_b^v = f \\ G(a,\ r^e) & \text{if } r^e = f,\ r_a^v = t,\ r_b^v = f \\ G(b,\ r^e) & \text{if } r^e = f,\ r_a^v = f,\ r_b^v = t \\ G(a,\ r^e) \bullet G(b,\ r^e) & \text{if } r^e = f,\ r_a^v = t,\ r_b^v = t \end{cases}$
$\Rightarrow$	$\{a,\ b\}$	$R = \begin{cases} G(a,\ r^e) + G(b,\ r^e) & \text{if } r^e = t,\ r_a^v = t,\ r_b^v = f \\ G(b,\ r^e) & \text{if } r^e = f,\ r_a^v = t,\ r_b^v = t \\ G(a,\ r^e) \bullet G(b,\ r^e) & \text{if } r^e = f,\ r_a^v = f,\ r_b^v = t \\ G(a,\ r^e) & \text{if } r^e = f,\ r_a^v = f,\ r_b^v = f \end{cases}$
$=$	$\{a,\ b\}$	$R = \begin{cases} \{modify = \langle a.element,\ a.property,\ r_b^v \rangle\} & \text{if } r^e = t,\ r_b^v = const \\ \{modify = \langle b.element,\ b.property,\ r_a^v \rangle\} & \text{if } r^e = t,\ r_a^v = const \\ \left\{ \begin{matrix} modify_1 = \langle a.element,\ a.property,\ r_b^v \rangle \\ \bullet \\ modify_2 = \langle b.element,\ b.property,\ r_a^v \rangle \end{matrix} \right\} & \text{if } r^e = t \\ \{modify = \langle a.element,\ a.property,\ \neq r_b^v \rangle\} & \text{if } r^e = f,\ r_b^v = const \\ \{modify = \langle b.element,\ b.property,\ \neq r_a^v \rangle\} & \text{if } r^e = f,\ r_a^v = const \\ \left\{ \begin{matrix} modify_1 = \langle a.element,\ a.property,\ \neq r_b^v \rangle \\ + \\ modify_2 = \langle b.element,\ b.property,\ \neq r_a^v \rangle \end{matrix} \right\} & \text{if } r^e = f \end{cases}$
$includes$	$\{a,\ b\}$	$R = \begin{cases} \{add = \langle a.element,\ a.property,\ r_b^v \rangle\} & \text{if } r^e = t \\ \{delete = \langle a.element,\ a.property,\ r_b^v \rangle\} & \text{if } r^e = f \end{cases}$
$\forall$	$\{a,\ b\}$	$R = \begin{cases} \left[\begin{matrix} \left\{ \bullet \bigcup_{i=1}^{n} delete_i = \langle a.element,\ a.property,\ r_{a_i}^v \mid r_{b_i}^v = f \rangle \right\} \\ + \\ \bullet \bigcup_{i=1}^{n} G(b_i \mid r_{b_i}^v = f,\ r^e) \end{matrix} \right] & \text{if } r^e = t \\ \left[\begin{matrix} \left\{ add = \langle a.element,\ a.property,\ r_{a_{n+1}}^v \mid r_{b_{n+1}}^v = f \rangle \right\} \\ + \\ + \bigcup G(b_i \mid r_{b_i}^v = t,\ r^e) \end{matrix} \right] & \text{if } r^e = f \end{cases}$
$\exists$	$\{a,\ b\}$	$R = \begin{cases} \left[\begin{matrix} \left\{ add = \langle a.element,\ a.property,\ r_{a_{n+1}}^v \mid r_{b_{n+1}}^v = t \rangle \right\} \\ + \\ + \bigcup G(b_i \mid r_{b_i}^v = f,\ r^e) \end{matrix} \right] & \text{if } r^e = t \\ \left[\begin{matrix} \left\{ \bullet \bigcup_{i=1}^{n} delete_i = \langle a.element,\ a.property,\ r_{a_i}^v \mid r_{b_i}^v = t \rangle \right\} \\ + \\ \bullet \bigcup_{i=1}^{n} G(b_i \mid r_{b_i}^v = t,\ r^e) \end{matrix} \right] & \text{if } r^e = f \end{cases}$

Benefits


1000
100
10
1
Small Rules Actions
Medium Size Rule Action
Large Rule Action
54
95
1789
1707
2438
2593
2809
5373
16255
33347
Model Size (# Model Elements)

Change Propagation: it's all about history

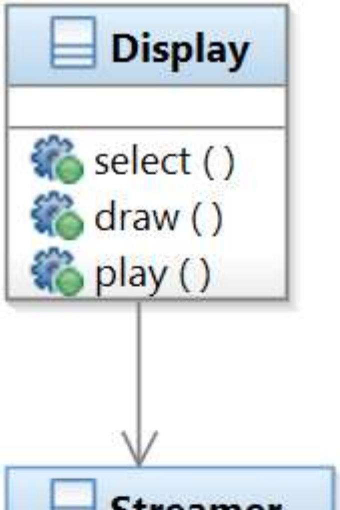
A Possible Dialog

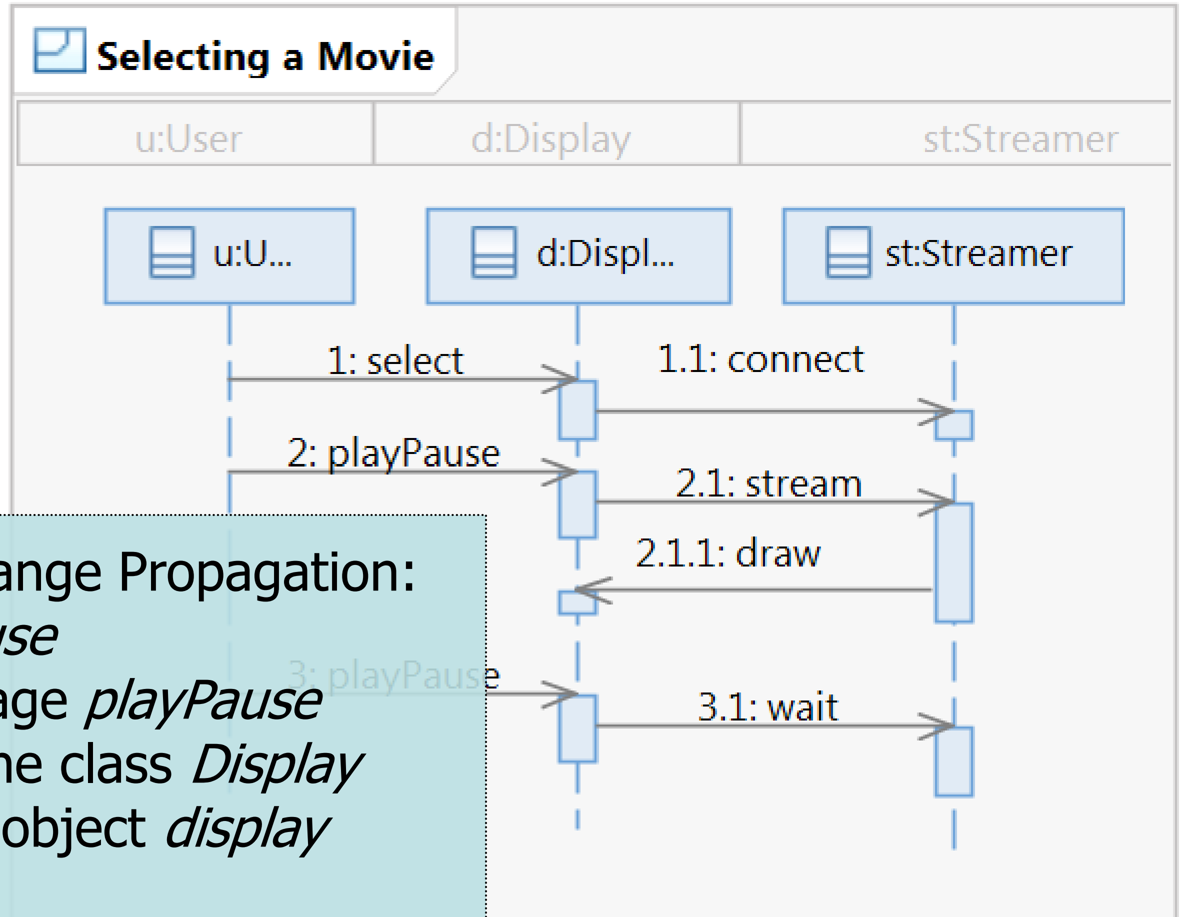
Designer

- Change the class diagram
  — HAL: can you help me propagate this change to the sequence diagram?

HAL

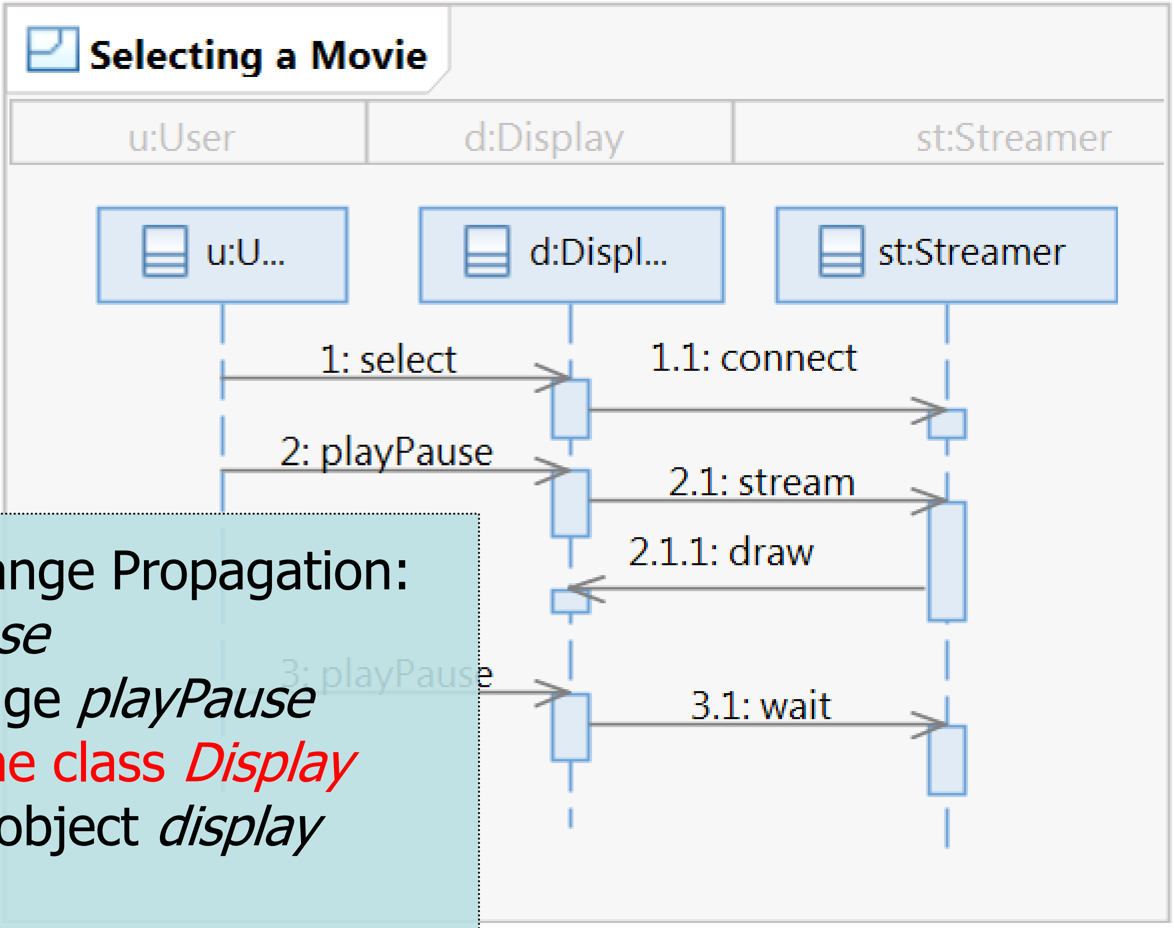
- Detects inconsistencies
- Computes repair alternatives
  — Assumption: no more changes to the class diagram





Alternative Locations for Change Propagation:

1) rename message *playPause*
2) Change receiver of message *playPause*
3) add a new operation to the class *Display*
4) change the ownership of object *display*
6) rename operation *select*
7) rename operation *play*
8) rename operation *draw*
9) delete message *playPause*



Alternative Locations for Change Propagation:

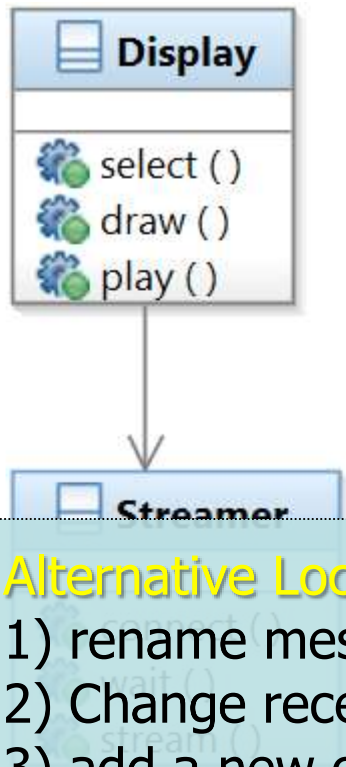
1) rename message *playPause*
2) Change receiver of message *playPause*
3) add a new operation to the class *Display*
4) change the ownership of object *display*
6) rename operation *select*
7) rename operation *playPause*
8) rename operation *draw*
9) delete message *playPause*

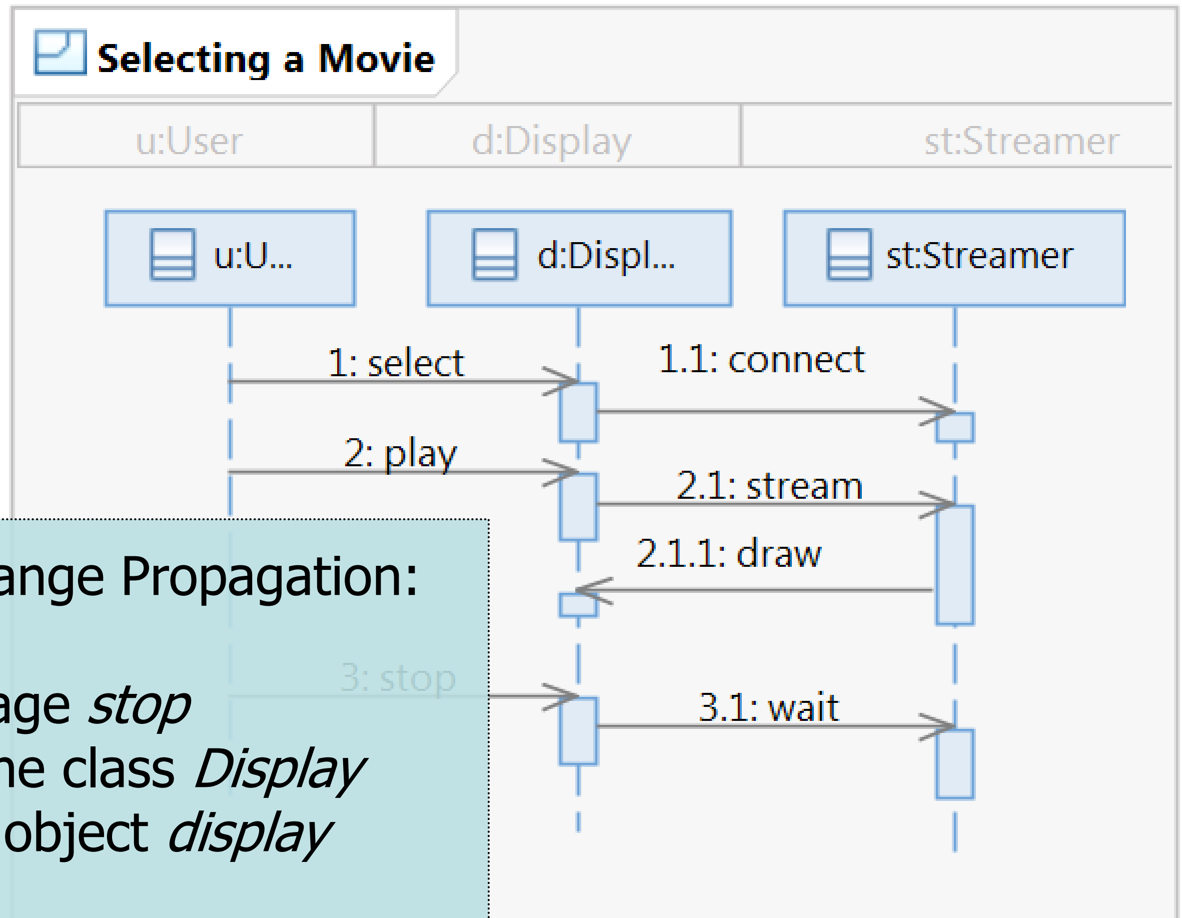
Tool

Execute repair (change propagation) that renames 1st message ‘playPause’ to ‘play’

Works in Reverse also. Rename message ‘playPause’ to ‘stop’.

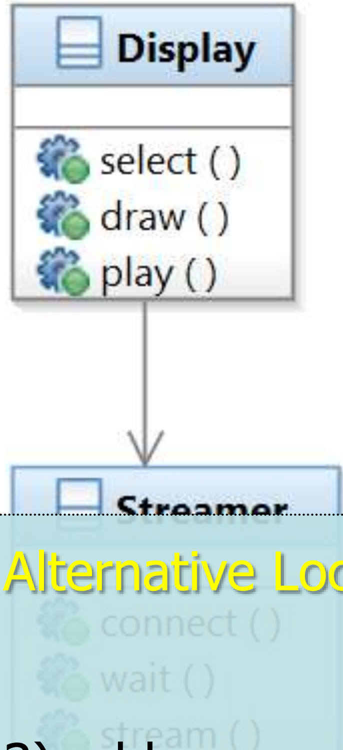
Show





Alternative Locations for Change Propagation:

1) rename message *stop*
2) Change receiver of message *stop*
3) add a new operation to the class *Display*
4) change the ownership of object *display*
6) rename operation *select*
7) rename operation *play*
8) rename operation *draw*
9) delete message *stop* (makes rule obsolete)



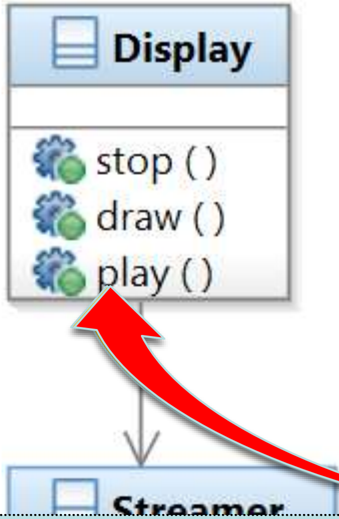
Alternative Locations for Change Propagation:

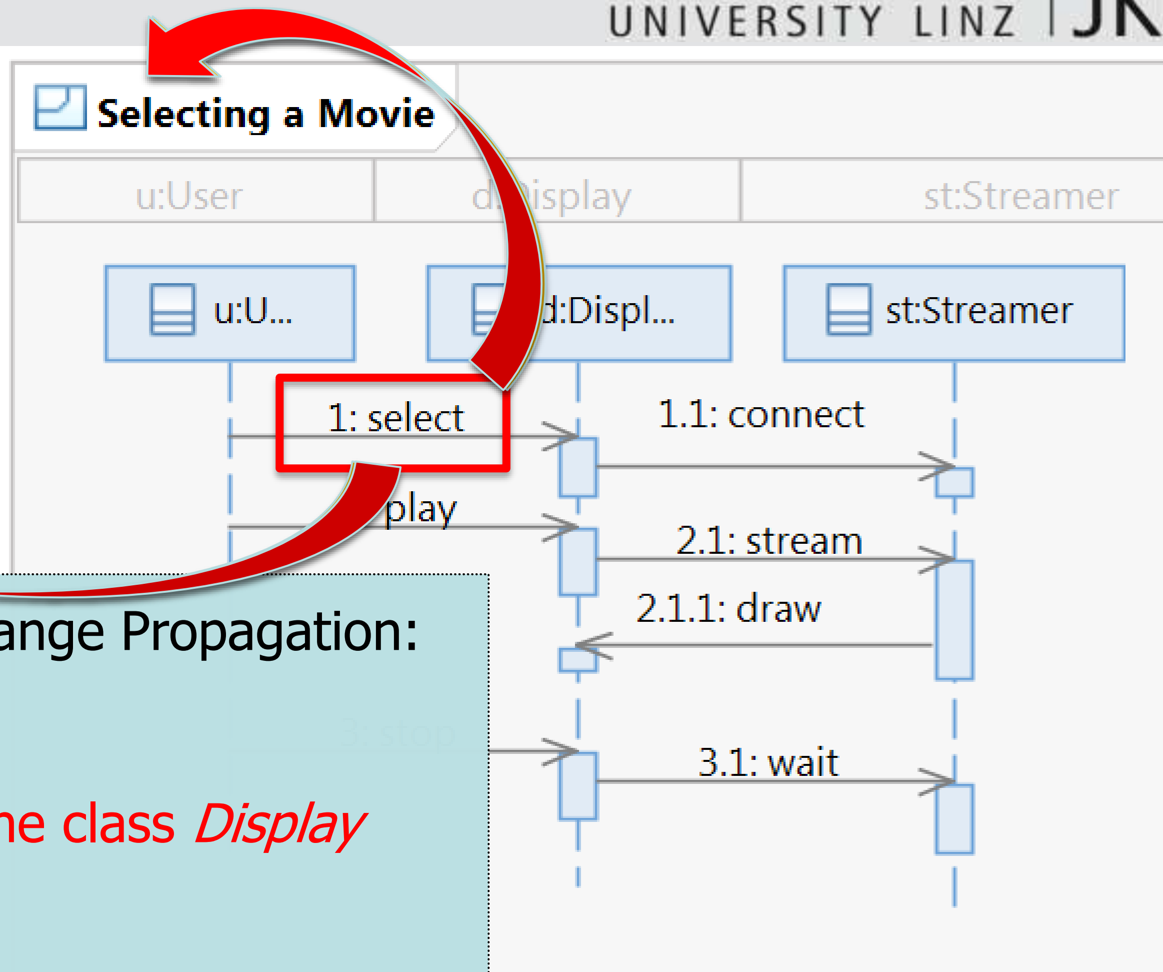
3) add a new operation to the class *Display*

6) rename operation *select*
7) rename operation *play*
8) rename operation *draw*

Tool

Show side effects of changing operations ‘select’, ‘play’, and ‘draw’





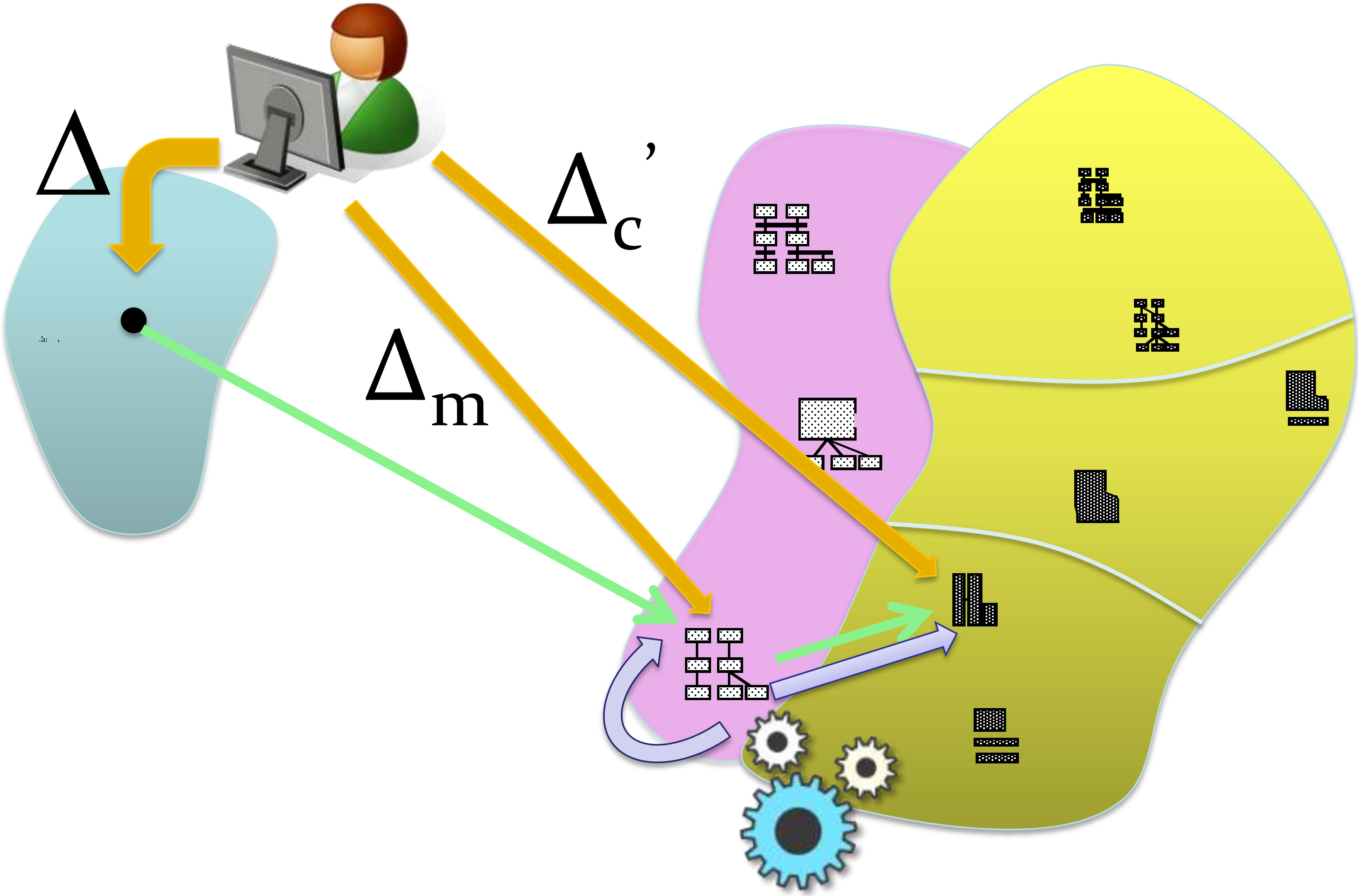
Alternative Locations for Change Propagation:

3) add a new operation to the class *Display*

6) rename operation *select*
7) rename operation *play*
8) rename operation *draw*

Maintaining the Model

JOHANNES KEPLER UNIVERSITY LINZ | JKU



Issues

We do not design automatically, we only propagate what is already known

A Change is only "propagetable" if there is a constraint that detects failure to propagate

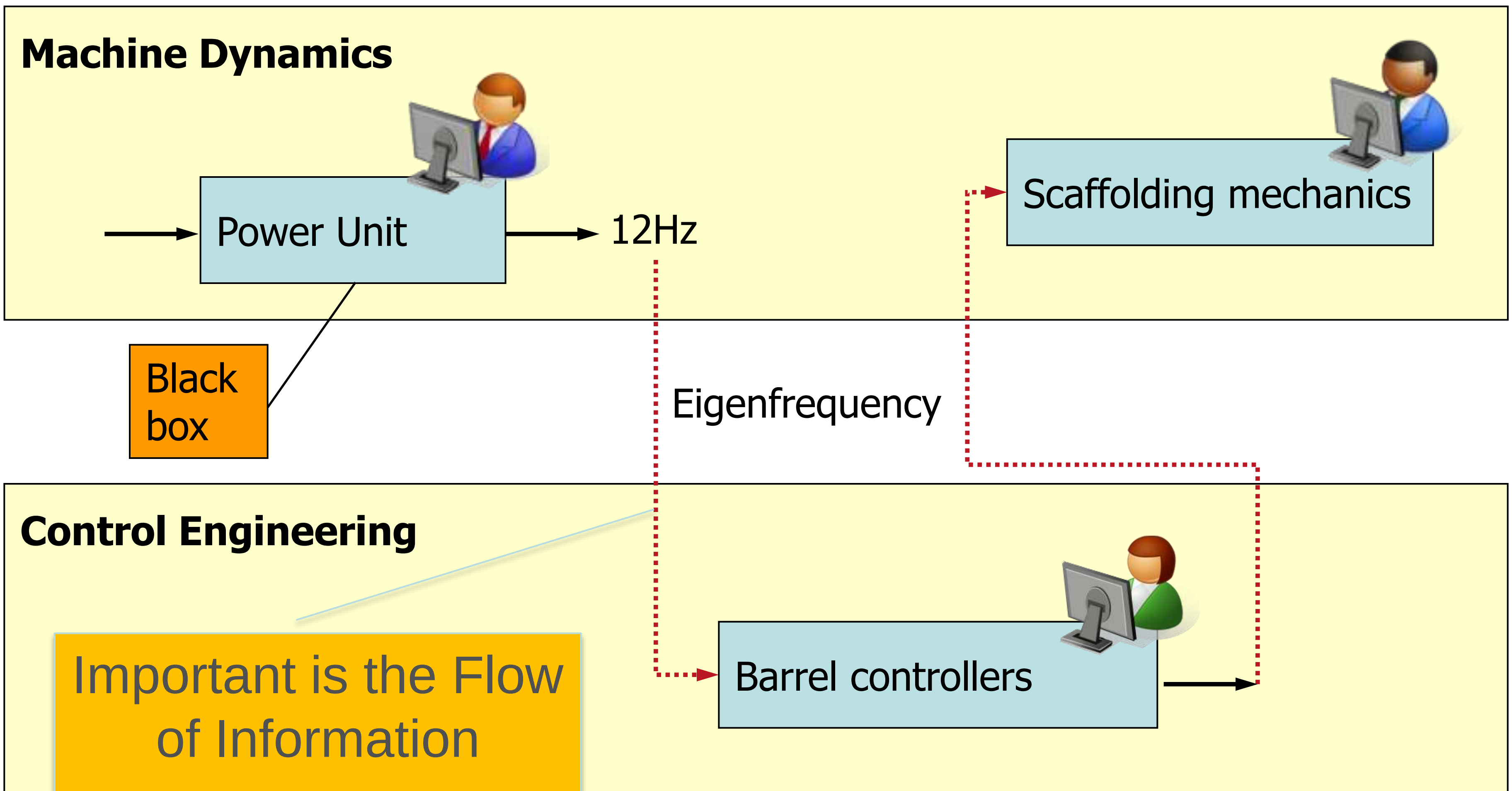
Models depict more than what you find in code

- Model maintenance cost will be higher
- $\Delta_c < \Delta_c' + \Delta_m$

Ongoing work

- Beyond design models
- Structural constraints vs. dynamic constraints
  - Invariant checking in code based on design constraints
- Applicable not just to software engineering
  - Integration with other disciplines

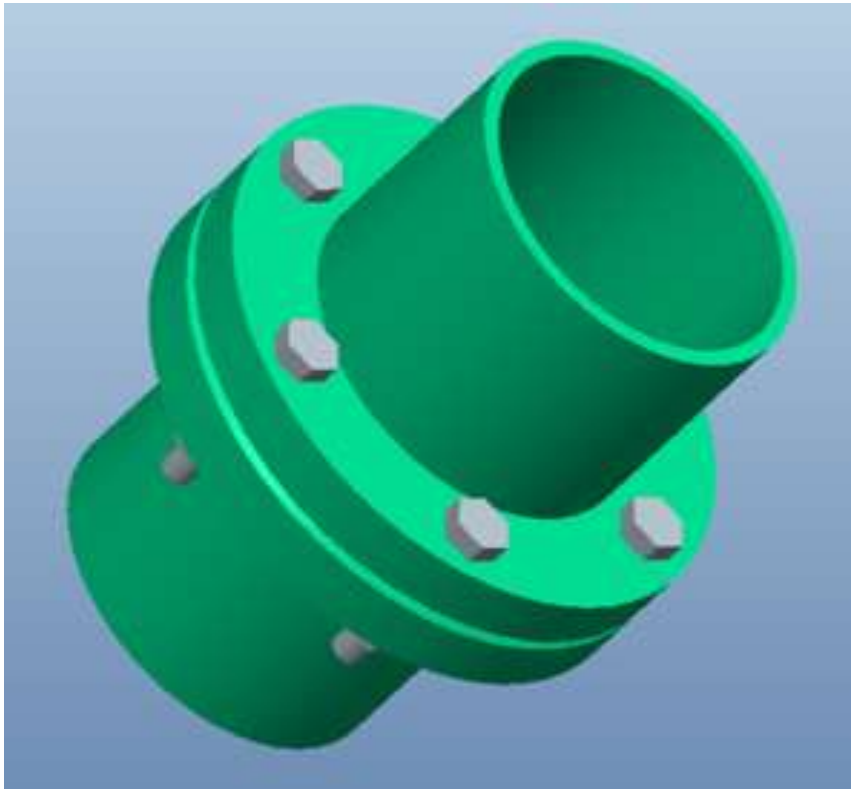
Inter-Disciplinary Collaboration



Example: Flange Connection

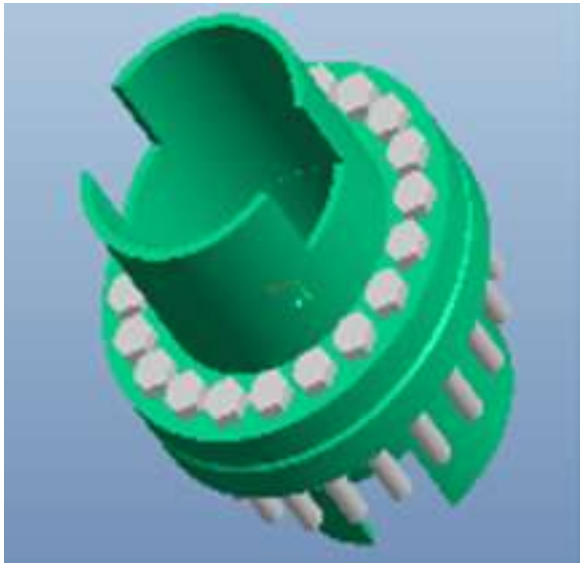
JOHANNES KEPLER UNIVERSITY LINZ | JKU

- The construction drawing is done in a CAD tool and the #screws calculation in Excel
- In the event of a change that influences the calculation of the number of screws



Example: Flange Connection

- In order to be able to assemble the flange connection another constraint has to be satisfied: BoltCircleDiameter * Pi > 1.4 * WrenchSize * #Screws



We gratefully acknowledge IBM and the Austrian FWF for funding this work under grant agreement P21321-N15

Contact me at alexander.egyed@jku.at

Johannes Kepler University, Linz (JKU)

http://www.sea.jku.at