MDD: the Good, the Bad and the Ugly

Steven Kelly
CTO, MetaCase
www.metacase.com
stevek@metacase.com

Markus Voelter
Independent/itemis AG
www.voelter.de
voelter@acm.org

Part 1

1 Goals

Greater Abstraction

… work on a higher level
… do more with less
… insulate from technology

Increase Value

… productivity
… quality
… responsiveness

Reduce lossiness

… domain → requirements
… requirements → design
… design → program

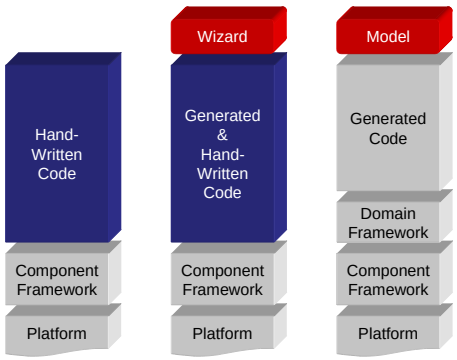
Goals

Not just any “MDD”

… domain-specific language
… domain-specific generator
… mostly made not bought

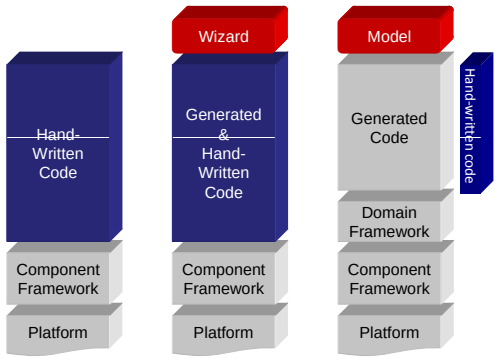
Goals

Generated ⇒ no edit



Goals

Generated ⇒ no edit



2 People, Process and Organization

Let people do what they're good at

Let 'em do what they're good at

MDD has several clearly defined Roles

Let 'em do what they're good at

Tech Experts

… evaluate technologies
… dig deep
… tune

… create templates
… spread knowledge

Let 'em do what they're good at

Language Designer

… works w/ domain expert abstractions, notations
… works w/ architect generators, interpreters

… requires "meta people"

Let 'em do what they're good at

App Developer

… cares about app domain
… uses DSLs + meta ware
… is isolated from technology

Let 'em do what they're good at

Flip side:

You actually need people who are good at this!

Worst practices: Initial conditions

Only Gurus Allowed

Only gurus build languages 4%

I'm smart & need no help 12%

Domain Users Programming?

Domain Users Programming?

Precision != Programming

Domain Users Programming?

Precision != Programming

Domain Users

Programmers

Programming = Precision + X

Domain Users Programming?

Precision

… Scientists
… Insurance Mathematicians
… Logisticians
… Medical Doctors
… Hardware Folks

Domain Users vs. Experts

Domain Users vs. Experts

Creating the Language

vs.

Using the Language

Domain Users vs. Experts

Creating: Domain Expert

Using: Domain User

Domain Users vs. Experts

Creating: Domain Expert

- … senior
- … complete
- … big picture
- … deep
- … precise, formal
- … guru

Domain Users vs. Experts

Using: Domain User

- … not senior
- … narrower
- … shallow

Worst practices: Initial conditions

Domain Dilettante

Insufficient understanding:
- Problem domain 17%
- Solution domain 5%

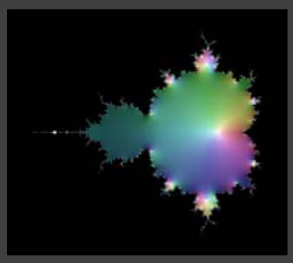
Compatible Organization

Never, ever

delegate to

interns!

Iterate!



Iterate!

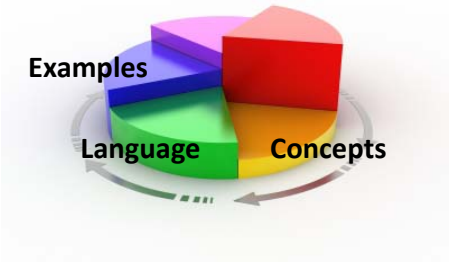
Waterfall is bad!

With or Without MDD

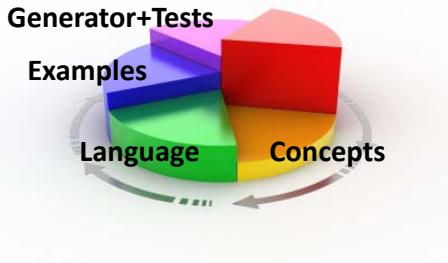
Iterate!

Iterate!

Iterate!



Iterate!



Iterate!

Worst practices: Initial conditions

Analysis Paralysis

Language must be known to be complete, fully implementable 8%

Iterate!

Instead:

Prototype!

… meaningful
… 4-8 person weeks
… incremental
… external help?

Compatible Organization

Compatible Organization

MDD requires **cross-project** work.

Compatible Organization

MDD requires **cross-project** work.

A strict **project-focused** organization does not work

Compatible Organization

Make **room & budget** for cross-cutting work.

Open Source-like Approach?

Compatible Organization

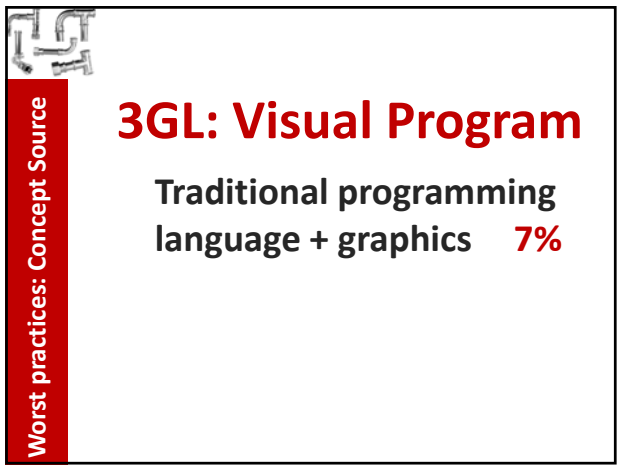
Suitability criteria

… **Mature** business area
… Work mostly **in-house**
… **Experts** available

3
Language Design

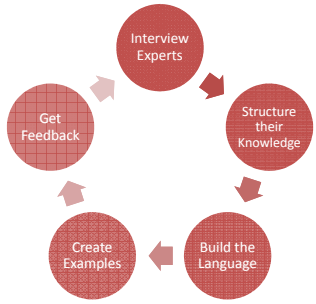
Sources for the Language

Worst practices: Concept Source
UML: Old Wineskins
Extend a large, general-purpose language 5%
+
!=


Worst practices: Concept Source
3GL: Visual Program
Traditional programming language + graphics 7%

Language Sources

Domain Analysis



Language Sources

Business DSLs

extracted from

Domain Knowledge	Domain Artifacts
• Interview Experts • Mine existing formalisms • Word, Excel	• Hardware Structure • Devices • User Interface

Language Sources

Technical DSLs

extracted from

… framework
… library
… pattern (language)
… architecture

Worst practices: Concept Source

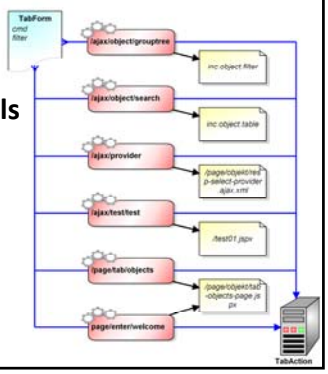
Code Components

The library is the language **32%**

Worst practices: Concept Source

Library is the Language

…low-level details
…duplication
…can't retarget
…level of abstraction?

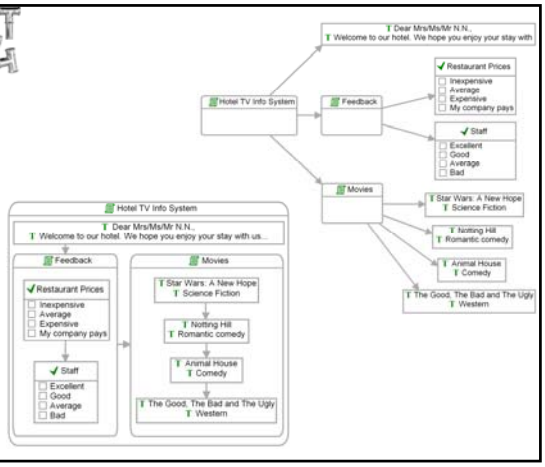


Worst practices: Concept Source

Tool: hammer ⇒ nails

Tool's technical limitations dictate language **14%**

Worst practices: Concept Source



Limit Expressiveness

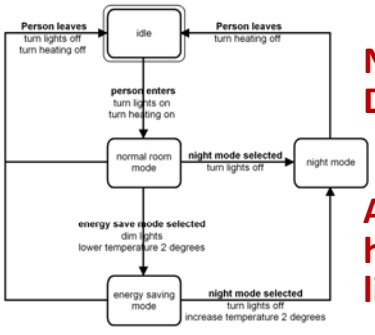
Worst practices: Concept Source

Too generic/specific

Too few/generic	21%
Too many/specific	8%
Language for 1 model	7%

Worst practices: Concept Source

Too few concepts



Not a DSL!

Add heating, lights?

Limit Expressiveness

Limit Expressiveness

Precision

precisely what

… facts
… declarative
… domain experts can!

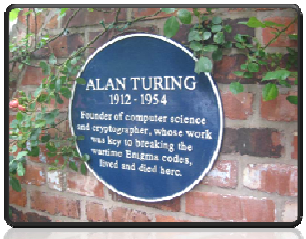
Limit Expressiveness

Algorithmic Completeness

formally how

… automation
… execution
… in model processors
… developers can!

Limit Expressiveness



Use a **3GL** if necessary
Generate APIs, Hooks

Documentation is still necessary

Documentation still necessary

The DSL and the "programs" are documentation.

Documentation still necessary

The DSL and the "programs" are documentation.

Not Quite!

Documentation still necessary

Language Definition is not a Teaching Tool!

Worst practices: No training

No training

"Everyone understands the language like me"

21%

Worst practices: No training

No training

"Everyone understands UML like me"

100 %

Documentation still necessary

Tutorials

- … Concepts
- … How to use Language
- … How to integrate manual code

Example-Driven!

Documentation still necessary

Language Definition captures the WHAT but not the WHY

Documentation still necessary

Rationales

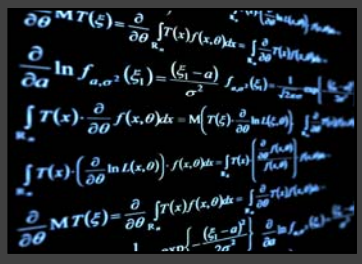
… why the concepts?
… why we generate what we generate
… target platform decisions and idioms

Documentation still necessary

Different Media

4 Notation

Notation, Notation, Notation



Notation, …

Domain Users
care deeply
about notation!

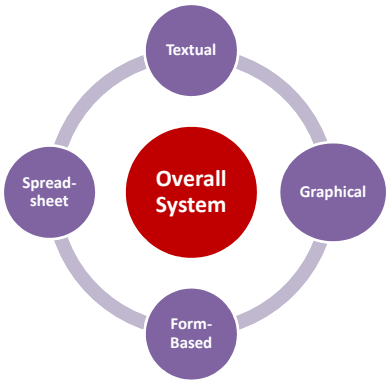
"UI" for the language

Notation, …

Textual
Semi-Graphical
Graphical
Form-Based
Matrices, Tables

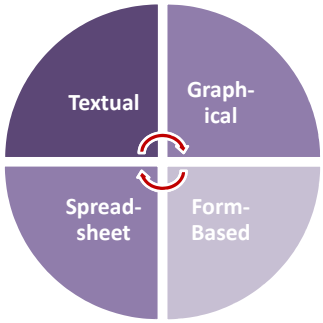
Notation, …

Parts



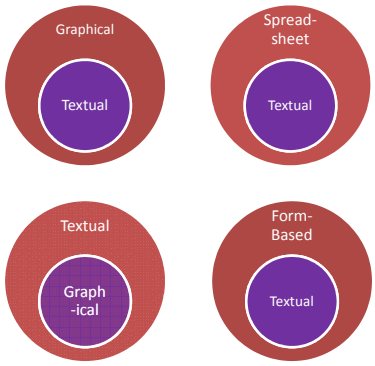
Notation, …

Convertible



Notation, …

Embedded

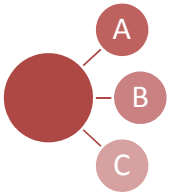


Notation, …

Graphical vs. Textual

Grapical vs. Textual

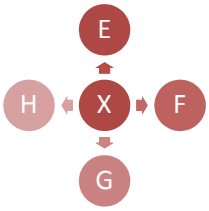
Graphical



Choices/Flow

Grapical vs. Textual

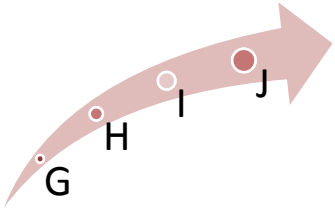
Graphical



Relationships

Grapical vs. Textual

Graphical



Timing

Grapical vs. Textual

Textual

In all other cases!

Grapical vs. Textual

Textual

Real simple…

…CVS/SVN Integration
… Diff/Merge
… Build automation
… Model Migration

Grapical vs. Textual

Text + Visualization

- Model-to-Model
- Automatic Layout
- Read-Only
- Drill-Down

Worst practice: notation

Predetermined Paradigm

Choosing wrong notation type because of blinkered view 7%

Graphical Syntax

Notation, …

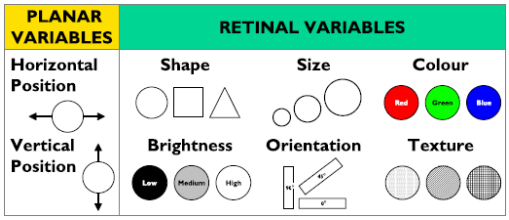
Symbol should have
1:1 mapping
to Domain concept

Notation, …

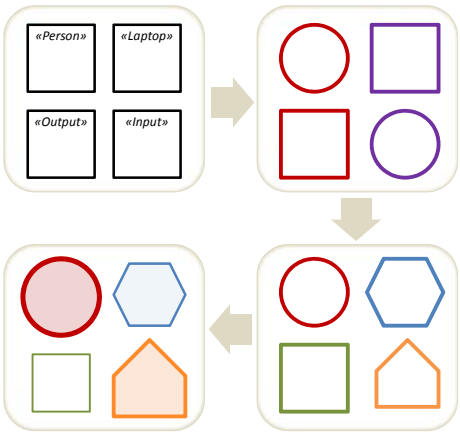
Symbol should
call to mind
Domain concept

Notation, …

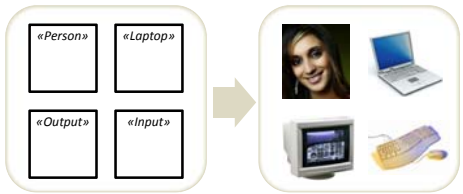
Symbols should use
full range
of visual variables



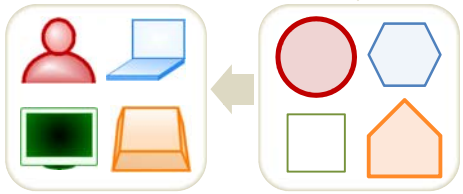
Notation, …



Notation, …



pictogram > geometric > photo



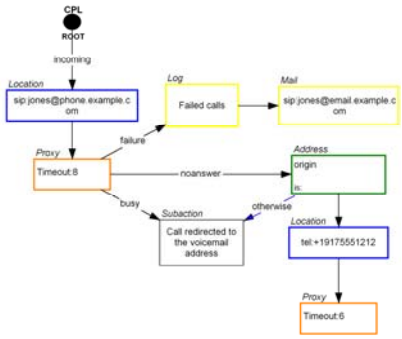
Worst practice: notation

Simplistic symbols

Too simple/similar 25%
Downright ugly 5%

Worst practice: notation

Color & label not enough



Notation, …

And by the way:

Notation, …

And by the way:

why not just use
text?

5 From Models To Code

Checks First and Separate

Checks first and separate

Concepts + Structure are not enough.

You need constraints.

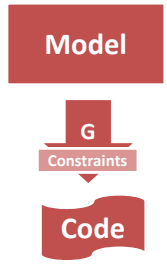
Boolean expressions that validate the model beyond structure.

Checks first and separate

Best: constraints as **data**
- part of language

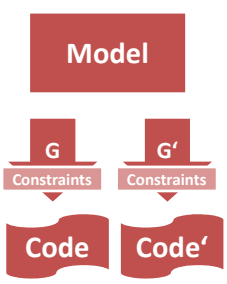
Worst: constraints as **code**
- requires impact analysis to scale
- but often needed

Checks first and separate



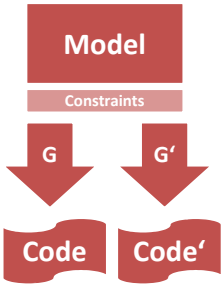
… complex

Checks first and separate



… complex
… duplication

Checks first and separate



separate phase first class citizens **much better.**

check as many as possible. **make it tight.**

Checks first and separate

different constraints at **different times** or for **different partitions** of the model

partition-local: **editor, on-save**
global: **batch, on-request**

Checks first and separate

Model
Constraints
T
Model‘
T
Model‘
G
Code

check early.
more semantics.
better messages.

Checks first and separate

Model
Constraints 1
T
Model‘
Constraints 2
T
Model‘
Constraints 3
G
Code

constraints 1 ok
implies
constraints 2 ok
implies
constraints 3 ok

Checks first and separate

ERROR
WARNING
INFO

Rich Domain Specific Platform

Rich Platform

Generated Application
Domain Frameworks
Libraries
Middleware
Operating System
Drivers

Rich Platform

Domain Frameworks

Libraries

Grown with the DSL!

Rich Platform

Extreme Case

Generated Application

populates

Domain Frameworks

Care about generated code

Andrew Vargas

Care about Generated Code

Generated Code a
Throwaway
Product?

Care about Generated Code

Generated Code a
Throwaway
Product?

Yes.
Can be regenerated.

Care about Generated Code

But:

must be…

… written (templates)
… understood
… debugged

Care about Generated Code

But:

must be…

… written (templates)
… understood
… debugged

if you don't generate 100%:

… extended
… programmed against

Care about Generated Code

Care!

… indent
… use good names
… document
… modularize

MDD

THE END.

…of part 1

MDD

come back tomorrow…

…for even better,
worse and uglier stuff.

Questions?

MDD: the Best, the Worst and the Ugliest

Markus Voelter
Independent/itemis AG
www.voelter.de
voelter@acm.org

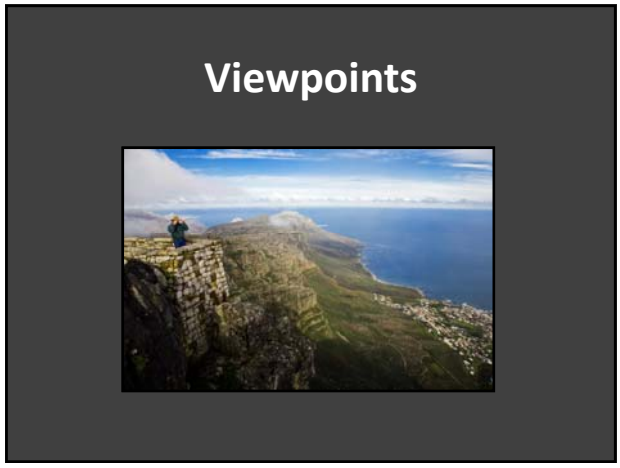
Steven Kelly
CTO, MetaCase
www.metacase.com
stevek@metacase.com

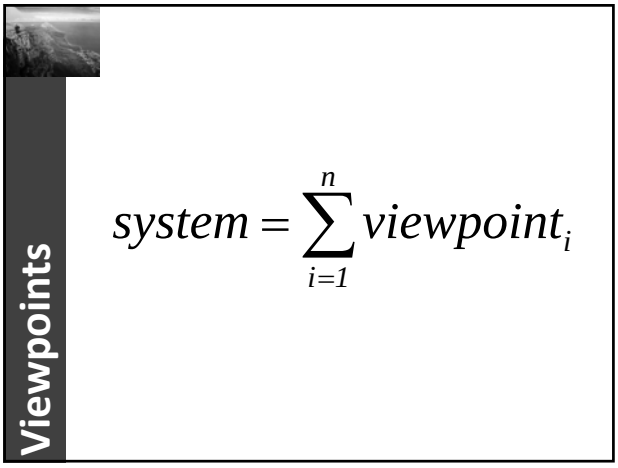
Part 2

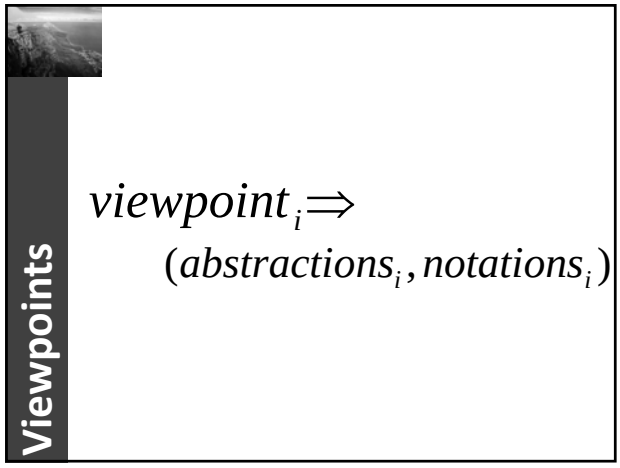
Quick Recap: Part 1

Quick Recap: Part 1

6 Breaking up Models


Viewpoints


Viewpoints
$system = \sum_{i=1}^{n} viewpoint_i$


Viewpoints
$viewpoint_i \Rightarrow (abstractions_i, notations_i)$

Viewpoints

Well-defined
Dependencies and
Connection Points

Viewpoints

Try to avoid
overlap and the need
for synchronization.

Viewpoints

Model Integration

Model Integration

Decomposition

- … Logical structure
- … Sub-models
- … Max 20 elements each
- … Split 7±2 ways / level

Model Integration

Summary Model

… Top-level overview
… Shows all sub-models
… Shows sub-model links

Model Integration

Side-by-side view

… 2+ models on screen
… Reduces memory load
… User chooses
… User positions

Model Integration

Cross-model links

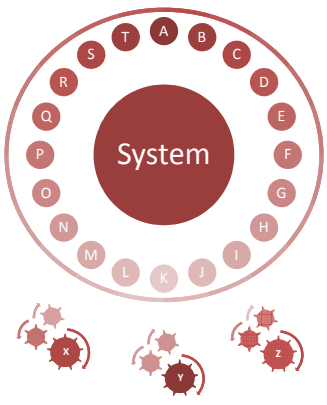
… Show referred objects
… Real object or pointer
… Use sparingly: coupling

Model Integration

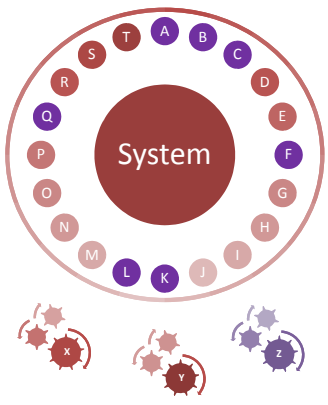
Configuration Viewpoint

Teamwork
Partitions
Viewpoints
Multiple Targets
Multiple Configurations

Model Integration



Model Integration



Partitioning

Partitioning

MDD tools do not **scale** arbitrarily!

- … performance
- … overview
- … search/find/organize
- … teamwork

Partitioning

Partitions

- … separate resources
- … != logical structure
- … units of check in/out
- … processable separately

Partitioning

cross ~~Partition~~ references

- … lazy
- … by proxy
- … by name w/ linker

Partitioning

Partition

- … not transparent
- … part of language design
- … referencable elements
- … “include path”

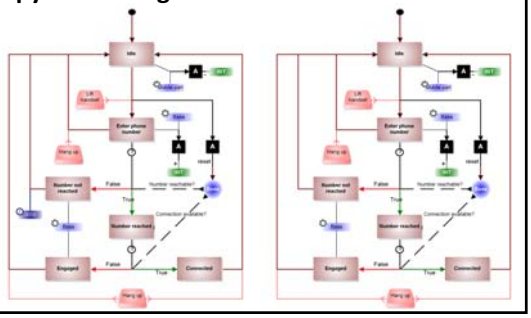
Partitioning

Annotation Models

Worst practices: ignore use

Ignoring Use Process 42%

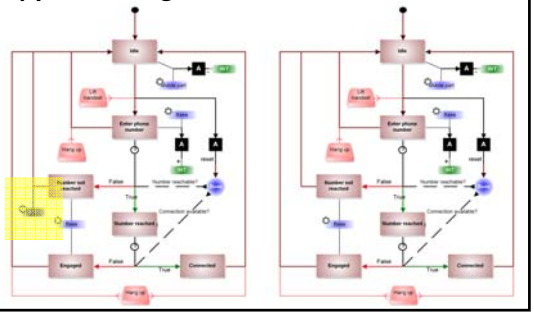
Copy entire diagram to handle minor variation?



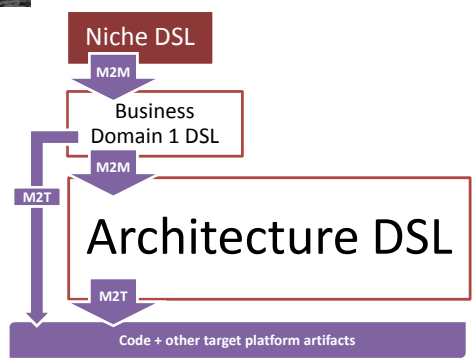
Worst practices: ignore use

Ignoring Use Process 42%

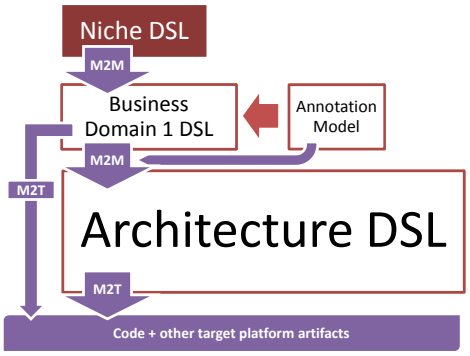
Copy entire diagram to handle minor variation?



Annotation Models



Annotation Models



Annotation Models

Annotation Model **references** elements in base model.

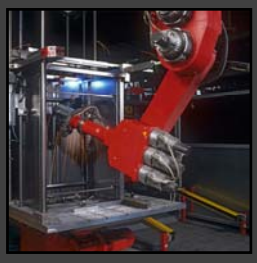
Transformation takes additional information into account.

Annotation Models

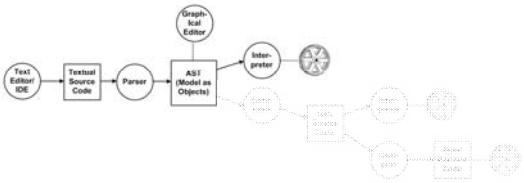
Make sure the **annotation model** only captures **exceptions** from the default in the generators.

7 Processing Models

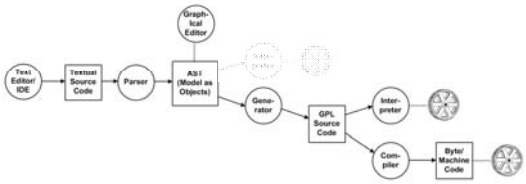
Interpretation vs. Generation



Interpretation



Generation



Generation

resulting code
can be easily
inspected

Generation

resulting code
can be easily
debugged

Generation

resulting code
can be
optimized
and more
efficient

Generation

Templates can be
derived
from existing code

Generation

work around
limitations
of target language

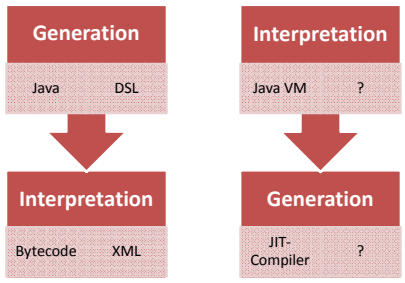
Generation

reuse
target runtime system

Interpretation

faster
turnaround
no regeneration
test
build
deploy

Combinations



Integrating Generated and Manually Written Code

Integrated Gen/Man Code

If at all possible,
Do not modify Generated Code!

Integrated Gen/Man Code

Protected Regions
are a bad idea because
generated code
is not a
throwaway
anymore.

Integrated Gen/Man Code

Protected Regions
… need to check in
… "sediment" of
generated code!

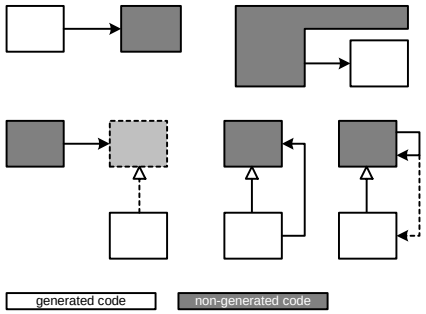
Integrated Gen/Man Code

Better Approach:
Hooks
in the generated code

Integrated Gen/Man Code

Better Approach:
Hooks
in the generated code
extension points, base class, abstract methods & subclassing, empty callback methods, delegate, implement interfaces, #include, reflection, AOP, design patterns, partial classes

Integrated Gen/Man Code



Care About Generators

Care about Generators

Generators

… important asset
… contain platform knowledge
… tend to grow more complex

Care about Generators

Care!

… modularize
… functions
… naming
… polymorphism
… aspects
… refactor

Care about Generators

Indent for the generator and then use a **beautifier** for the **generated code.**

Care about Generators

Indent for the generator
and then use a
beautifier
for the
generated code.

except if you use a langauage
with semantic whitespace!

Care about Generators

Good Platform,
Simpler Generator,
Less Care.

Simplifying Generators

Simplifying Generators

Reducing generator
complexity
is important.

Simplifying Generators

Reducing generator
complexity
is important.

Separation of Concerns
is the way to go.

Simplifying Generators

Instead of putting
boilerplate
or **complex code**
in generators as fixed text
refactor it into the
domain framework

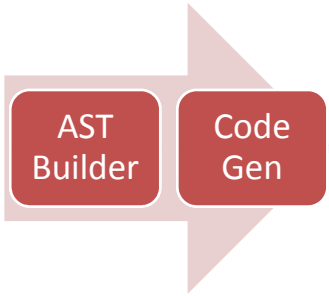
Simplifying Generators

Instead of putting
complex logic
into the generators
use O-O principles &
metaprogramming to
keep them flexible

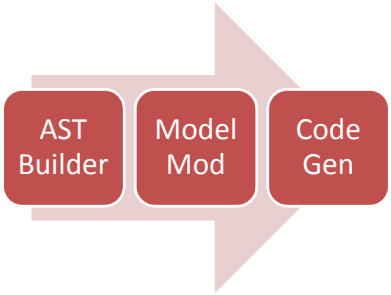
Simplifying Generators

Instead of putting
complex logic
into the generators
put it into an M2M
that runs
before code gen.

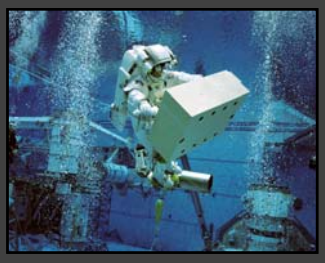
Simplifying Generators



Simplifying Generators



Model-2-Model For Simulation and Proof



M2M for Simulation and Proof

Many
useful formalisms
already exist.

M2M for Simulation and Proof

Many
useful formalisms
already exist.

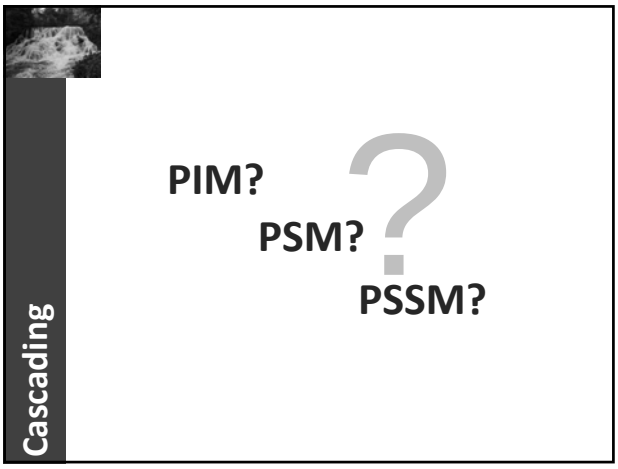
Simulation
Proofs
Properties

M2M for Simulation and Proof

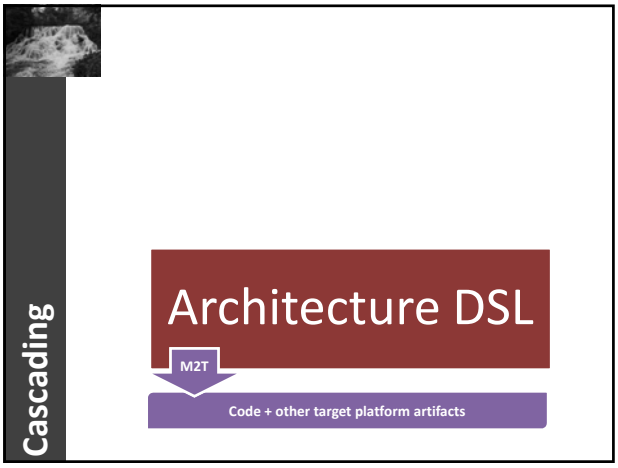
Use an **M2M**
for this if possible.

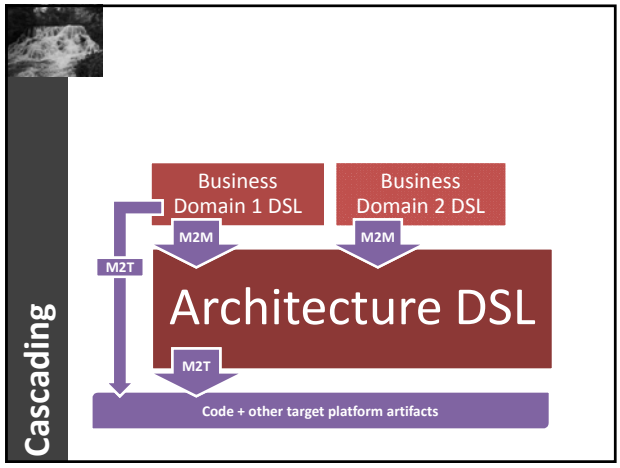
Often the input is **XML**
so you actually
"**generate code**"

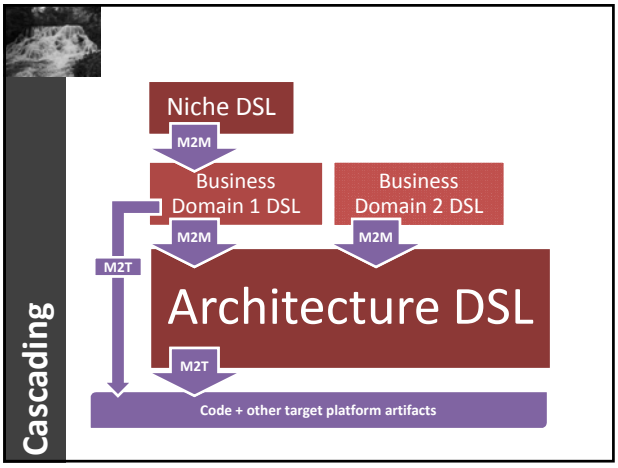
Cascading

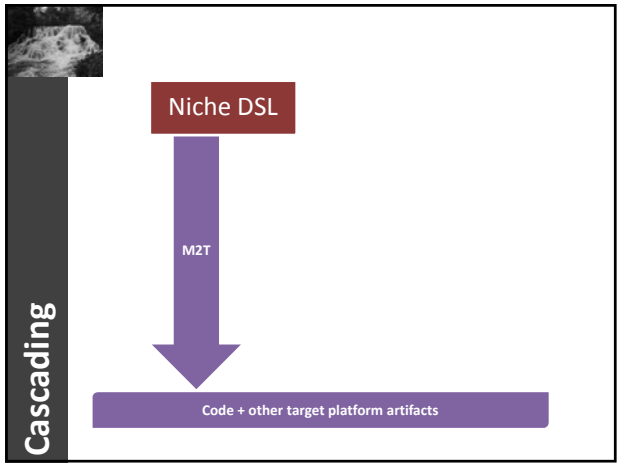

Cascading
PIM?
PSM?
PSSM?

Cascading
Bottom Up
Works Better!


Cascading
Architecture DSL
M2T
Code + other target platform artifacts


Cascading
Business Domain 1 DSL
Business Domain 2 DSL
M2M
M2M
M2T
Architecture DSL
M2T
Code + other target platform artifacts


Cascading
Niche DSL
M2M
Business Domain 1 DSL
Business Domain 2 DSL
M2M
M2M
M2T
Architecture DSL
M2T
Code + other target platform artifacts


Cascading
Niche DSL
M2T
Code + other target platform artifacts

Don’t forget Testing

Don’t Forget Testing

Limited Expressiveness.
Reduced Need For Tests.

Don’t Forget Testing

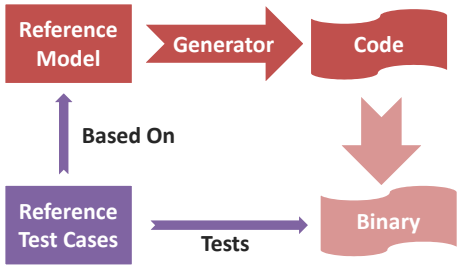
Constraint Checks.
A Form of Test.

Don’t Forget Testing

Testing Generators

Don’t Forget Testing

Testing Generators

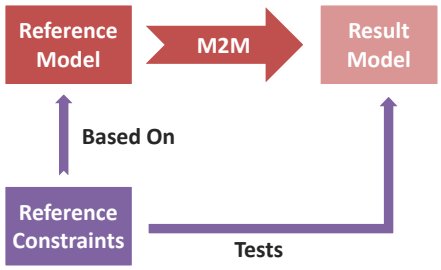


Don't Forget Testing

Testing Transformations

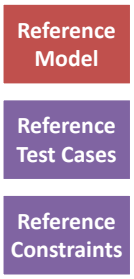
Don't Forget Testing

Testing Transformations



Don't Forget Testing

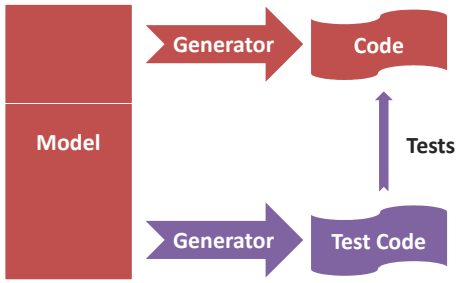
Testing Metaware



… maintained!
… by metaware developers

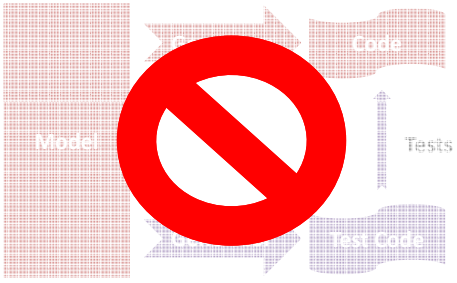
Don't Forget Testing

Generating tests



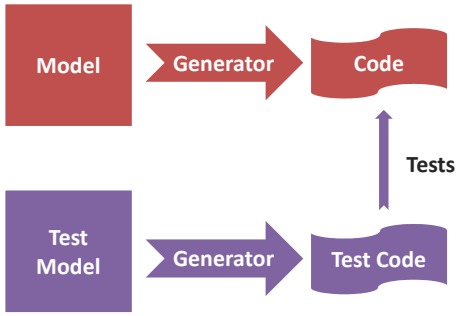
Don't Forget Testing

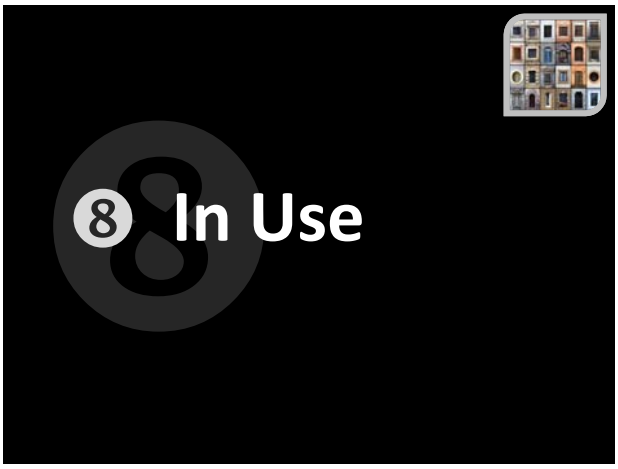
Self-Fulfilling ….!



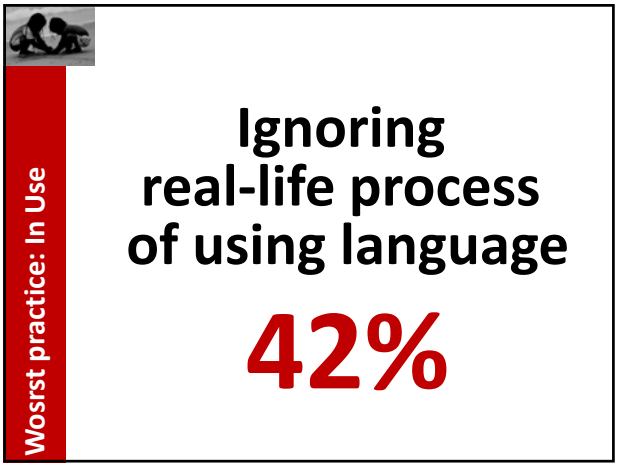
Don't Forget Testing

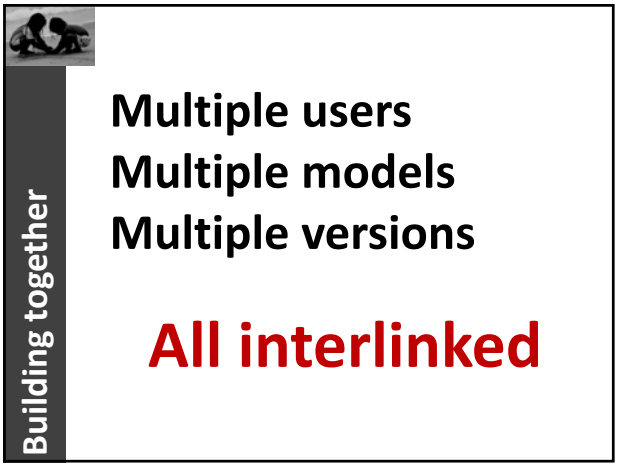
Generating tests




8 In Use

Building Together
TRAIL-SEEKER.COM


Wosrst practice: In Use
Ignoring
real-life process
of using language
42%


Building together
Multiple users
Multiple models
Multiple versions
All interlinked

Wosrst practice: In Use

Trying to model like you coded

modeling != coding

Wosrst practice: In Use

Trying to model like you coded

modeling != coding?

modeling ≅ coding

modeling != coding

Building together

Same old problems but new material

Old solutions don't apply

... new processes

... new tools

modeling != coding

Building together

Diff + merge:

Text easy, models hard

Multi-user editing:

Text hard, models easy

Co-Evolve Language and Concepts

Co-Evolve Language & Domain

Understanding the Domain

Building The Language

Co-Evolve Language & Domain

Understanding the Domain

Building The Language

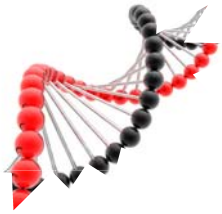
Co-Evolve Language & Domain

Building a language requires **Formalization**

Co-Evolve Language & Domain

Building a language requires **Formalization** requires you to **think** and **decide** about the domain.

Co-Evolve Language & Domain



requires frequent **Evolution…**

Co-Evolve Language & Domain

… hence flexible, agile **Tooling!**

Worst practices: In Use

Letting language
stagnate
after successful start

37%

Evolution

Evolution

What to do with existing
models
if the
language
changes?

Evolution

Might Require…

… config management
… version tag in models
… change tracking
… migration M2M

Evolution

Think about…

… backward compatibility
… deprecation
… instrumentation
… viewpoints + partitions

Evolution

THE END.

...of it all.

Questions?