MD* Best Practices

Markus Voelter

Independent/itemis

www.voelter.de
voelter@acm.org

Architect
Consultant
Coach
Trainer

Software
Architecture

Domain
Specific
Languages

Model
Driven
Development

Software
Product Line
Engineering

Conference Presenter

OOPSLA • JAOO • AOSD • EclipseCon • OOP • JAX

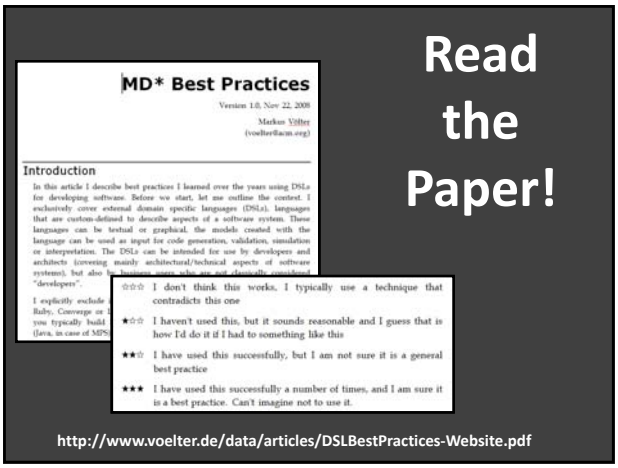
Podcaster

software engineering radio

go to:
www.voelter.de

Based in
Stuttgart
Germany

Glider Pilot


MD* Best Practices
Version 1.0, Nov 22, 2008
Markus Völter
(voelter@acm.org)
Introduction
☆☆☆ I don't think this works, I typically use a technique that contradicts this one
★☆☆ I haven't used this, but it sounds reasonable and I guess that is how I'd do it if I had to something like this
★★☆ I have used this successfully, but I am not sure it is a general best practice
★★★ I have used this successfully a number of times, and I am sure it is a best practice. Can't imagine not to use it.
Read the Paper!
http://www.voelter.de/data/articles/DSLBestPractices-Website.pdf

Based on actual practical experience

Benchmarked by suitability for use in today's projects

1 Designing the Language

Limit Expressiveness

Limit Expressiveness

Limit Expressiveness

Configuration

select
…from config space

- … Properties
- … Feature Models

Limit Expressiveness

Customization

compose
… creatively

- … Vocabulary
- … Sentences
- … Box and Line

Limit Expressiveness

Precision

precisely what

… facts
… declarative
… domain experts can!

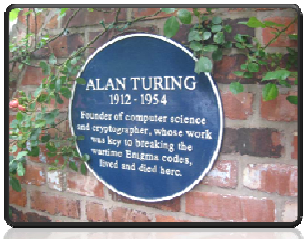
Limit Expressiveness

Algorithmic Completeness

formally how

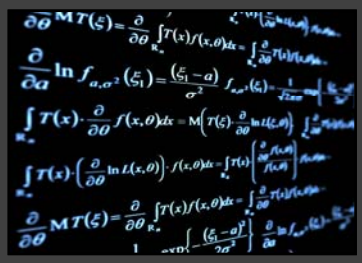
… automation
… execution
… in model processors
… developers can!

Limit Expressiveness



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

Notation, Notation, Notation



Notation, …

Domain Users
care deeply
about notation!

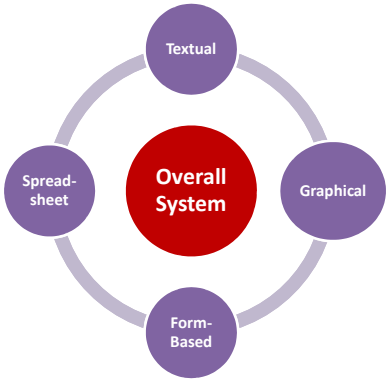
„UI“ for the language

Notation, …

Textual
Graphical
Form-Based
Spreadsheet

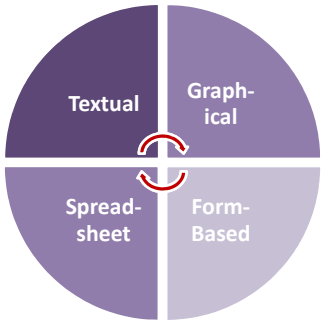
Parts

Notation, ...



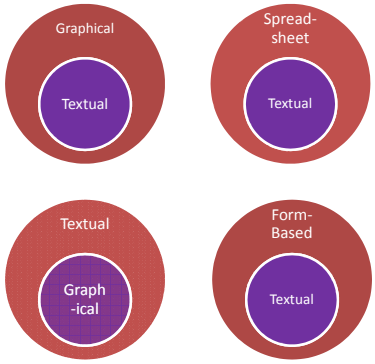
Convertible

Notation, ...



Embedded

Notation, ...



Notation, ...

Graphical vs. Textual

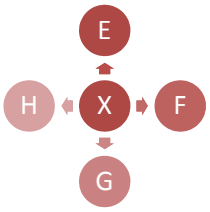
Graphical

Grapical vs. Textual

Sequence/Flow

Grapical vs. Textual

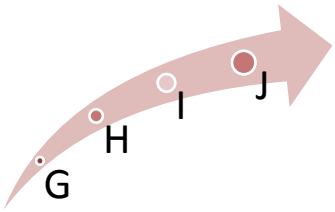
Graphical



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

Viewpoints

Viewpoints

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

Viewpoints

$$viewpoint_i \Rightarrow (abstractions_i, notations_i)$$

Viewpoints

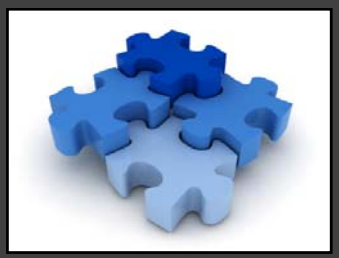
Well-defined Dependencies and Connection Points

Viewpoints

Try to avoid overlap and the need for synchronization.

Viewpoints

Partitioning



Partitioning

MD* tools do not **scale** arbitrarily!

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

The Fallacy of Generic Languages

Fallacy of Generic Lang.

Fallacy of Generic Lang.

You can model everything with

Fallacy of Generic Lang.

You can model everything with

somehow!

Fallacy of Generic Lang.

Problem

Shoehorning domain abstractions into the generic language

Fallacy of Generic Lang.

Problem

Sidetracked by existing abstractions and notations

Fallacy of Generic Lang.

Notations/ Abstractions extensible via Profiles

Very Limited Tool Support!

Fallacy of Generic Lang.

Meta Model Complexity!

Fallacy of Generic Lang.

But don't reinvent the wheel either.

Fallacy of Generic Lang.

Where are **standards** useful?

Fallacy of Generic Lang.

People have to learn **underlying concepts** anyway.

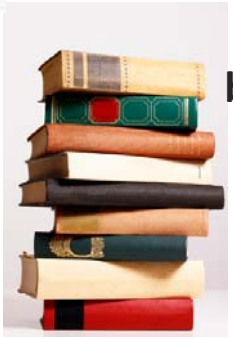
Fallacy of Generic Lang.

Is UML with a profile still a **standard language**?

Fallacy of Generic Lang.

On which **meta level** do I want to standardize? M2 (UML), M3 (MOF)?

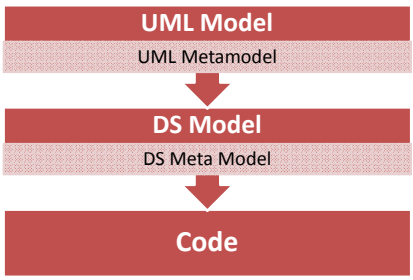
Fallacy of Generic Lang.



Isn't a **DSL** based on **MOF** as „standard" as a profile based on UML?

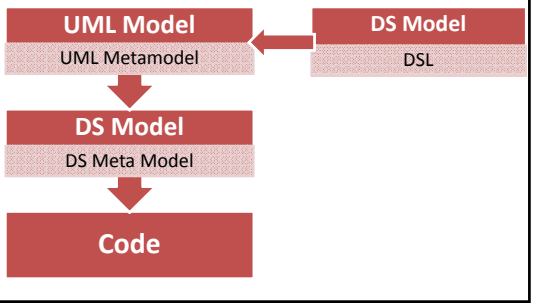
Fallacy of Generic Lang.

Introduce an **intermediate** language



Fallacy of Generic Lang.

separate **viewpoints** UML + DSL



Learn from 3GLs

Learn from 3GLs

DSL != 3GL

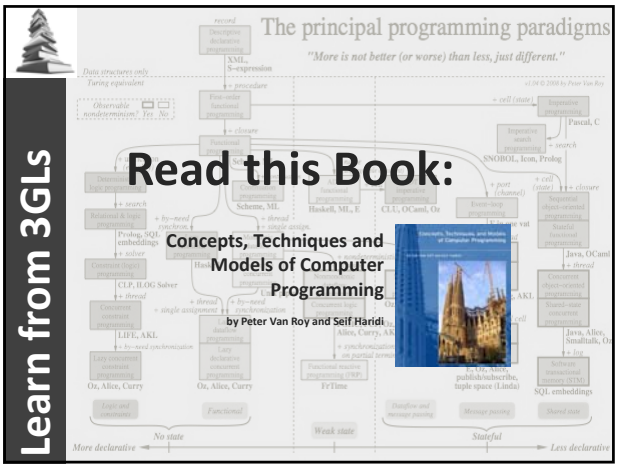
But:

Learn from 3GLs

We can learn:

- Namespaces
- Scoping
- Visibility
- Type Systems
- „Abstract"
- Specialization …

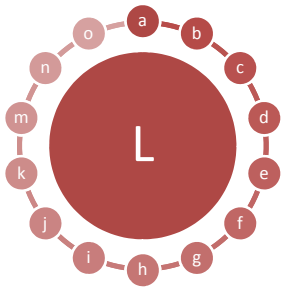
Learn from 3GLs



Who are 1st Class Citizens?

Who are 1st Class Citizens

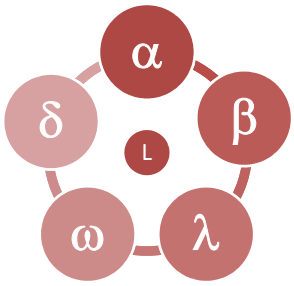
Big Language



with many concepts!

Who are 1st Class Citizens

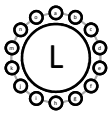
Small Language



with few, but
poweful concepts!

Who are 1st Class Citizens

Big Language

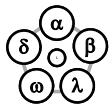


Easier for Business Users
Concepts easy to find

COBOL style

Who are 1st Class Citizens

Small Language



Technical DSLs
Concepts harder to find
More expressive

Lisp Style

Reuse and Variations

Do not
mix
the two styles!

Teamwork Support

Teamwork Support

Versioning
Lock
Check in/out
Diff/compare
Merge
Branch
Tag

Teamwork Support

Versioning
Lock
Check in/out
Diff/compare
On the Merge
Level of the Branch
Concrete Syntax! Tag

Teamwork Support

Versioning
Lock
Check in/out
Together Diff/compare
with manually Merge
written Source! Branch
Tag

Teamwork Support

Repository

vs.

File-Based

Teamwork Support

Repository

… Element-Specific
… Real-Time
… Often good for Business DSLs

Teamwork Support

File-Based

… like SCMs
… integrates well with manually written code
… Technical DSLs

Teamwork Support

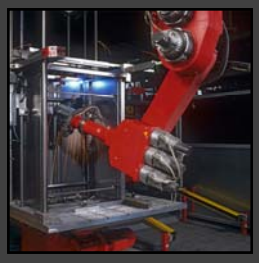
File-Based

… integrates very well with (real) textual DSLs

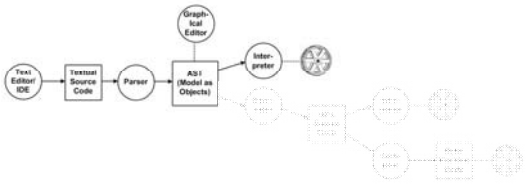
Teamwork Support

2 Processing the Models

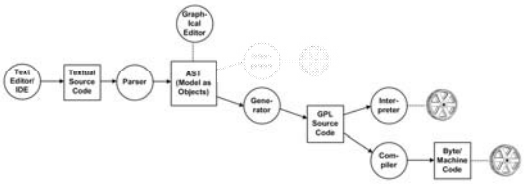
Interpretation vs. Generation



Interpretation



Generation



Generation

resulting code
can be easily
inspected

Generation

resulting code
can be easily
debugged

Interpretation vs. Generation

Generation

resulting code
can be
optimized
and more
efficient

Interpretation vs. Generation

Generation

Templates can be
derived
from existing code

Interpretation vs. Generation

Generation

work around
limitations
of target language

Interpretation vs. Generation

Generation

no changes
to target environment

(leaves no trace)

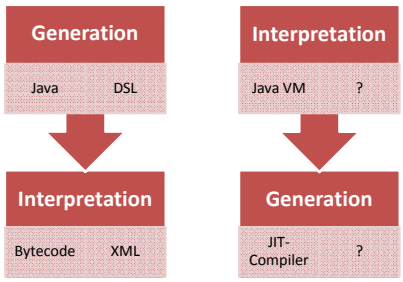
Interpretation vs. Generation

Interpretation

faster
turnaround
no regeneration
test
build
deploy

Interpretation vs. Generation

Combinations



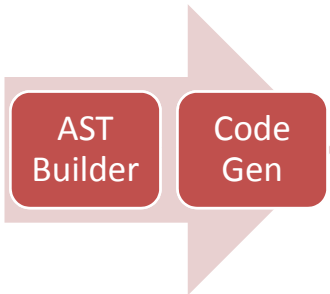
Model-2-Model To Simplify Generators

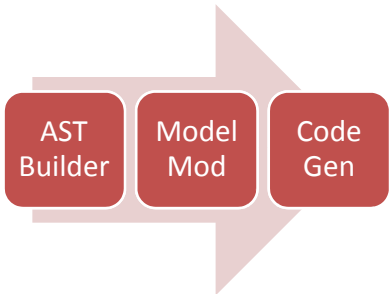
Reducing template **complexity** is important.

Reducing tempalte **complexity** is important.

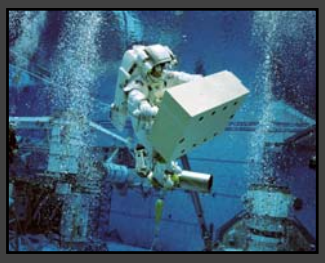
Separation of Concerns is the way to go.

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





Model-2-Model 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.

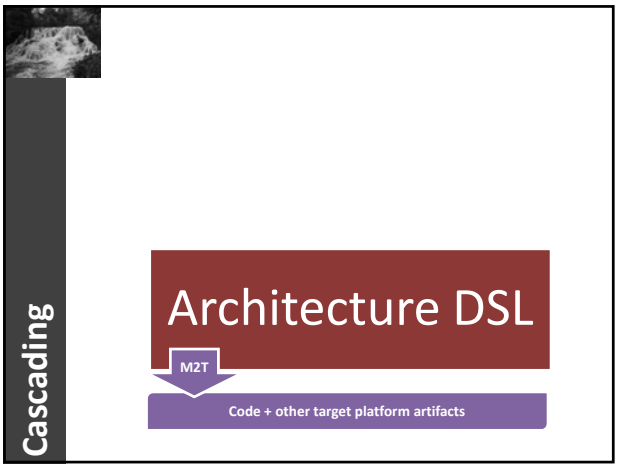
Often the input is **XML**
so you actually
„**generate code**“

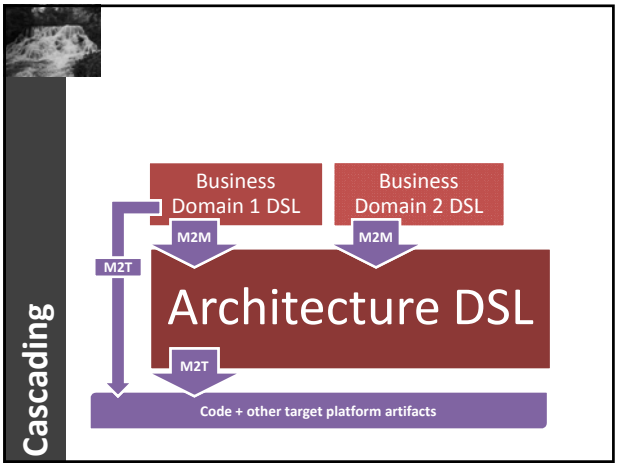
Cascading

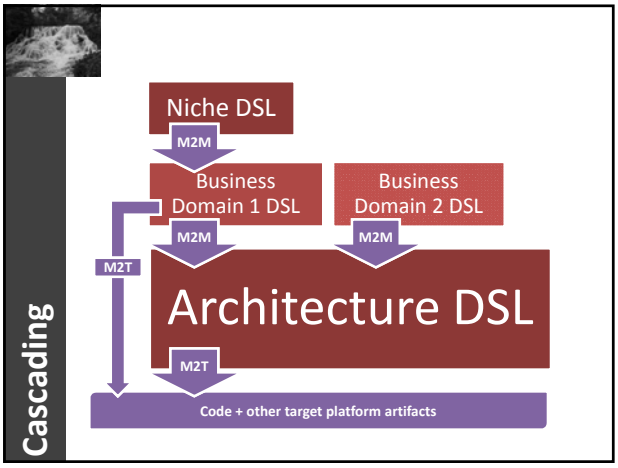
PIM?
PSM?
PSSM?

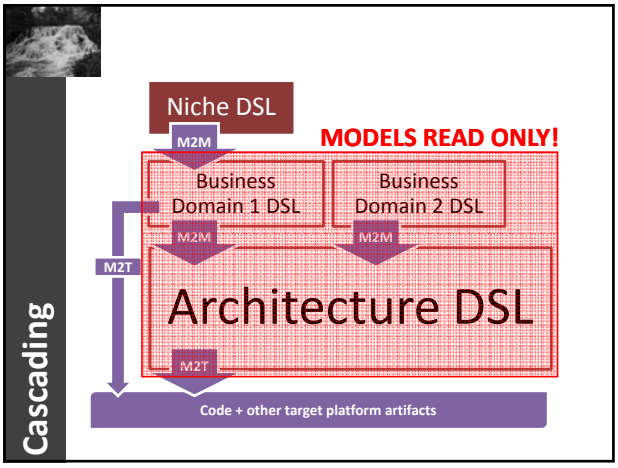
?

Cascading
Bottom Up
Works Better!


Cascading
Architecture DSL
M2T
Code + other target platform artifacts

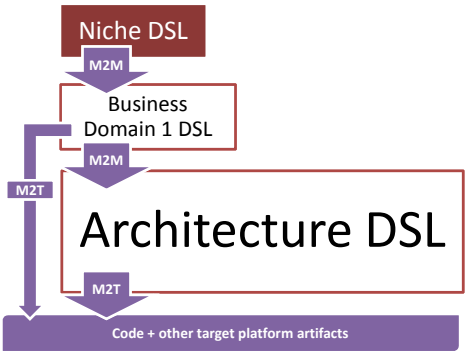

Cascading
Business Domain 1 DSL
Business Domain 2 DSL
M2M
M2M
M2T
Architecture DSL
M2T
Code + other target platform artifacts


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

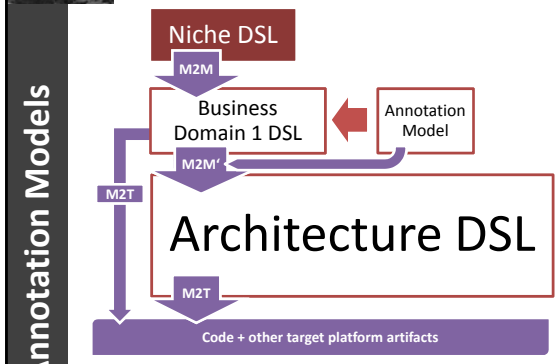

Cascading
Niche DSL
M2M
MODELS READ ONLY!
Business Domain 1 DSL
Business Domain 2 DSL
M2M
M2M
M2T
Architecture DSL
M2T
Code + other target platform artifacts

Annotation Models

Annotation Models



Annotation Models



Annotation Models

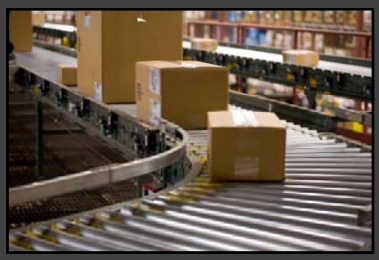
Annotation Model
references
elements in base model.

Transformation
takes additional information into account.

Annotation Models

Make sure the
annotation model
only captures
Exceptions
from the default in the templates.

Classify Behaviour



Classify Behaviour

Action Semantics Languages ?

useful?

Classify Behaviour

Action Semantics Languages ?

useful! But…

Classify Behaviour

Classify!

… state based
… business rules
… mathematics

Classify Behaviour

Classify!

… state based
… business rules
… mathematics

… or a specific DSL

Classify Behaviour

Classify!

… state based
… business rules
… mathematics

… or a specific DSL
… 3GL code

Don‘t forget Testing

Don‘t Forget Testing

Limited Expressiveness.
Reduced Need For Tests.

Don't Forget Testing

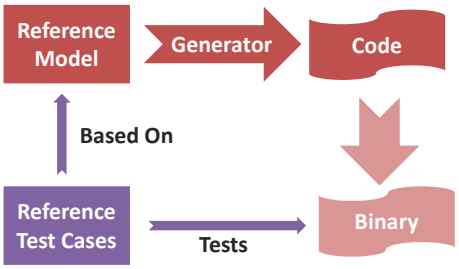
Constraint Checks.
A Form of Test.

Don't Forget Testing

Testing Generators

Don't Forget Testing

Testing Generators

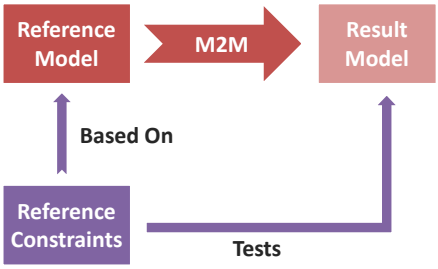


Don't Forget Testing

Testing Transformations

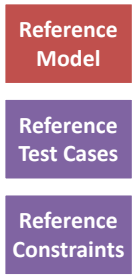
Don't Forget Testing

Testing Transformations



Don't Forget Testing

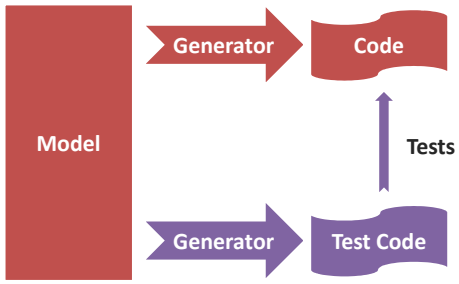
Testing Metware



… maintaned!
… by metaware developers

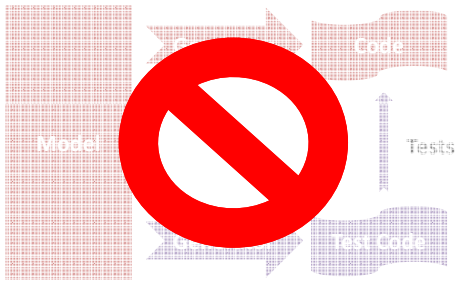
Don't Forget Testing

Self-Fulfilling ….!



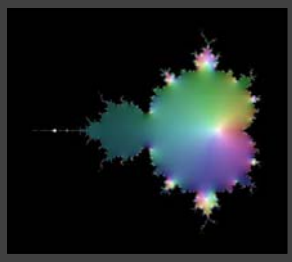
Don't Forget Testing

Self-Fulfilling ….!

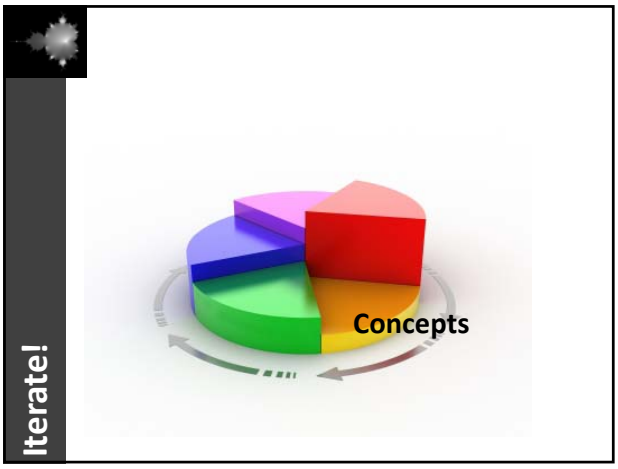


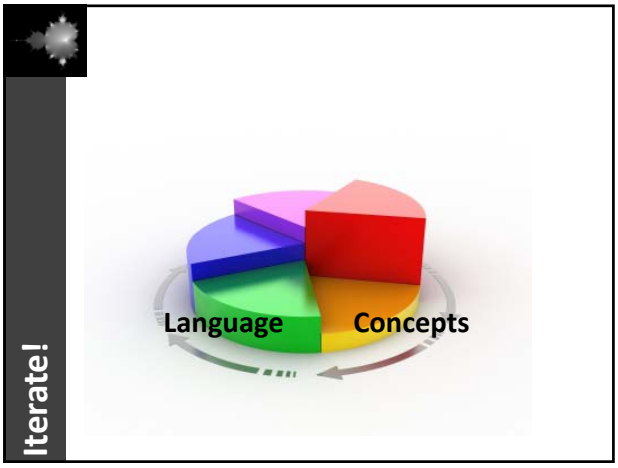
3 Process the Organization

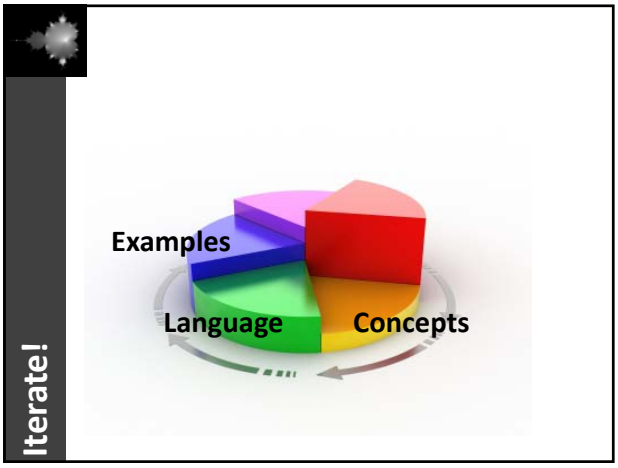
Iterate!

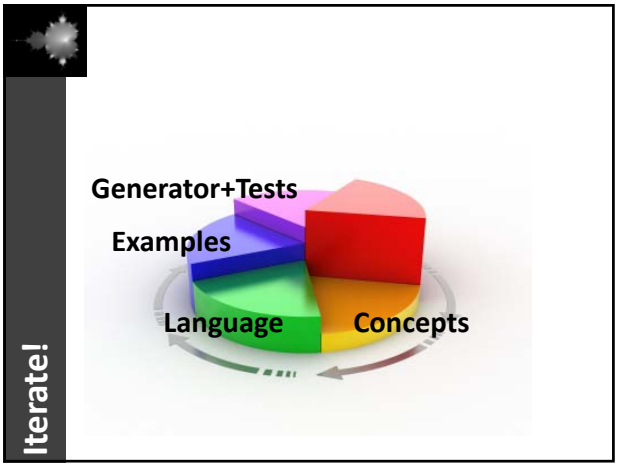


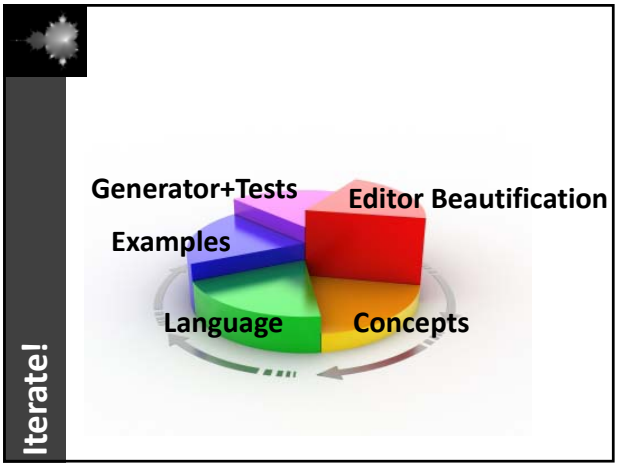
Iterate!
Waterfall is bad!
With or Without MD*


Iterate!
Concepts


Iterate!
Language
Concepts


Iterate!
Examples
Language
Concepts


Iterate!
Generator+Tests
Examples
Language
Concepts


Iterate!
Generator+Tests
Editor Beautification
Examples
Language
Concepts

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!**

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

Reviews

Reviews

In most cases, people can still make mistakes.

Reviews

DSL programs are more concise, so reviews are more efficient.

Reviews

Repeated „Mistakes“? Add constraint check.

Reviews

Repeated „Mistakes“?
Add constraint check.

Or maybe the language is wrong and it is **not a mistake?**

Let people do what they‘re good at

Let‘em do what they‘re good at

MD* 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!

Metaware as Product

Metaware as Product

Product:

… release schedule
… incremental developmt
… requirements mgt.
… issue tracking
… documentation
… support staff

Metaware as Product

Exhange People

… make meta developers develop real apps
… let app developers develop metaware

Compatible Organization

Compatible Organization

MD* requires
cross-project
work.

Compatible Organization

MD* requires
cross-project
work.

A strict project-focused
organization does
not work

Compatible Organization

Make
room & budget
for cross-cutting
work.

Open Source?

THE END.
Thank you.
Questions?

.coordinates

web www.voelter.de
email voelter@acm.org
skype schogglad

xing http://www.xing.com/profile/Markus_Voelter
linkedin http://www.linkedin.com/pub/0/377/a31

© 2009 Markus Völter