

Architecture As Language



Markus Voelter

www.voelter.de
voelter@acm.org

itemis


Andreas Graf

www.itemis.de
graf@itemis.de

itemis

About


itemis



1

What is a language?

INFORMAL




Set of well-defined terms



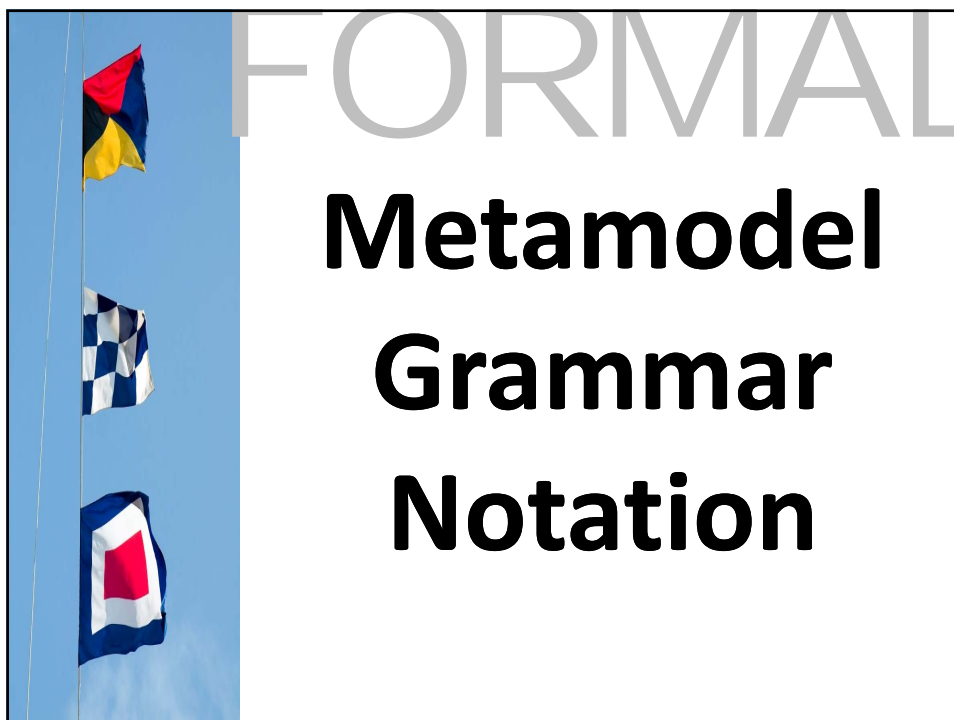
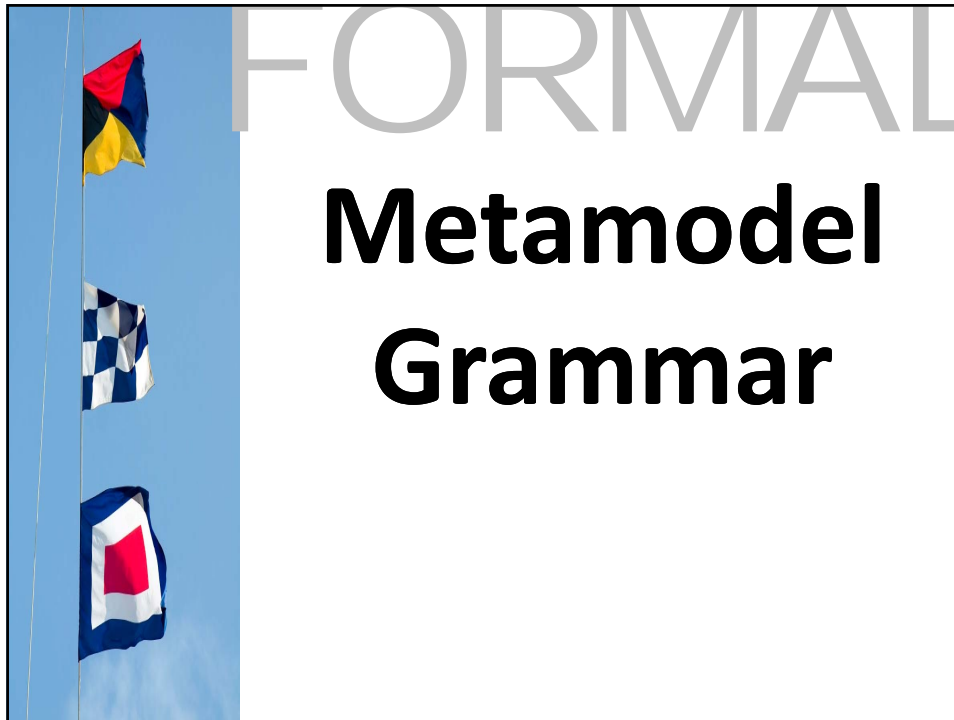
INFORMAL

**Stakeholders
agree on
meaning**



FORMAL

Metamodel



A DSL is a **focussed, processable language** for describing a specific **concern** when building a system in a specific **domain**. The **abstractions** and **notations** used are natural/suitable for the **stakeholders** who specify that particular concern.



2

Architecture DSLs



Architecture

DSL



**As you
understand
and develop
your
Architecture...**

**Develop a language to
express it!**



Language resembles architectural concepts



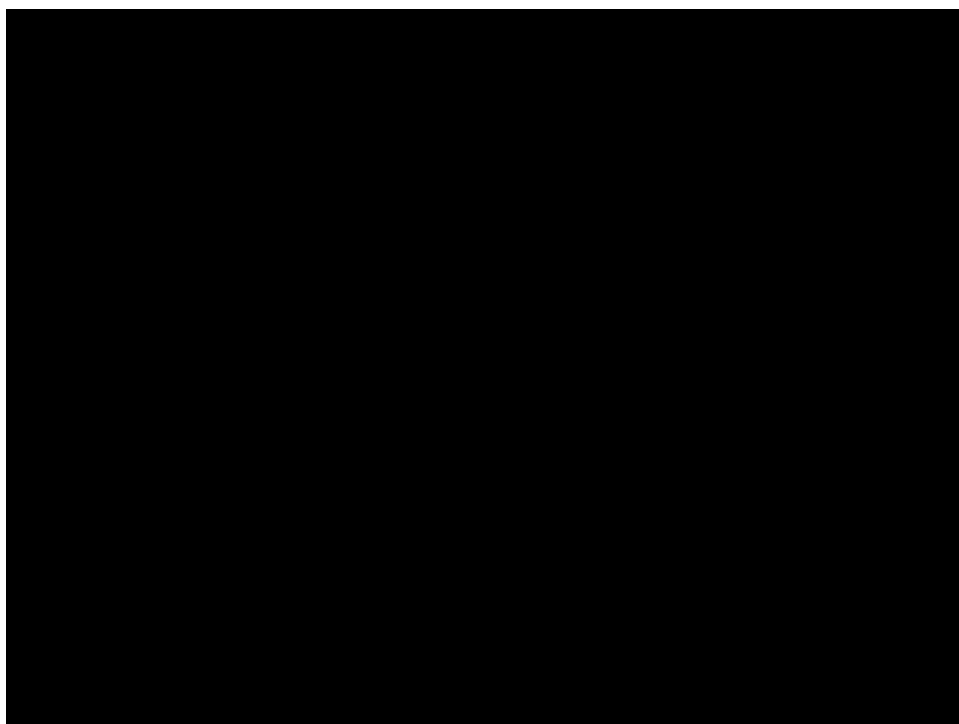
**We express the
application(s) with
the language.**

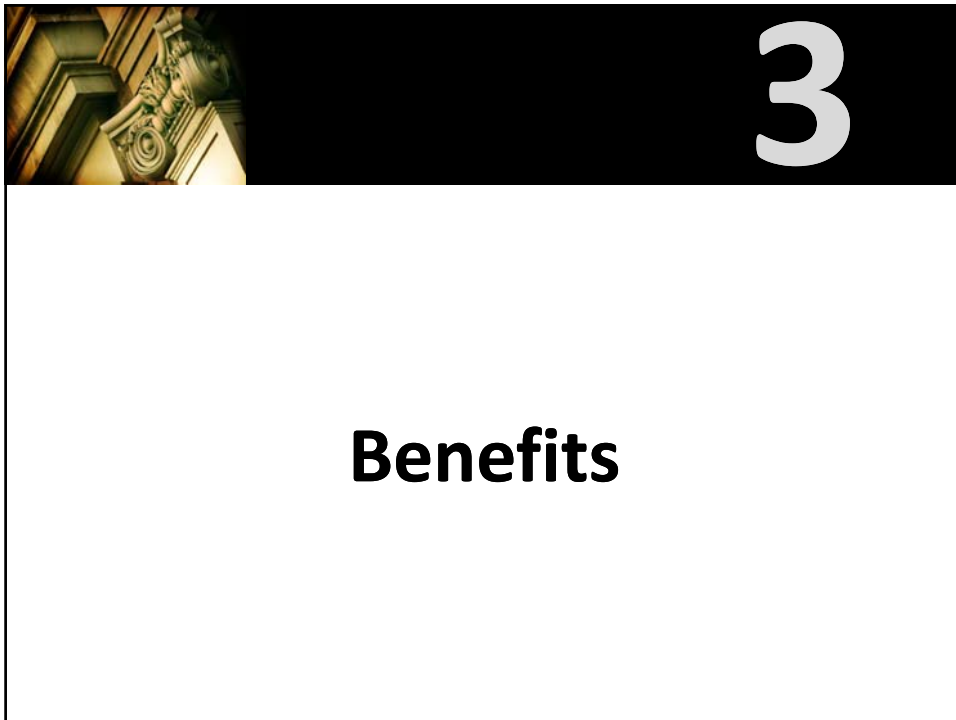


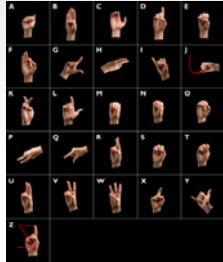
DEMO I



An architectural DSL for
embedded systems







Unambiguous Vocabulary

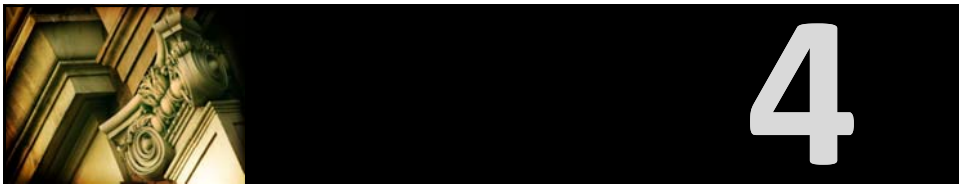
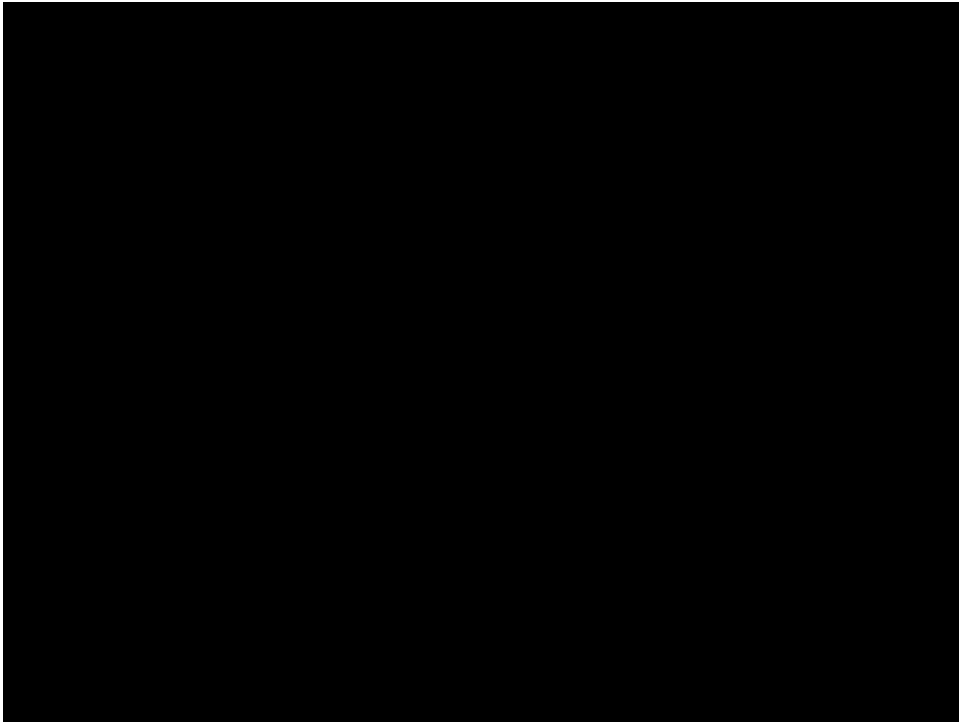
Concepts independent from Technology



**Programming Model can
be defined based on
Conceptual Arcitecture**

**Architecture „executable“
(i.e. more than rules and docs)**





4

Why Textual?

A decorative header bar at the top of the lower section features a photograph of classical architectural columns on the left and a large, white, bold number '4' on a black background on the right. Below this header is a white rectangular area with a thin black border. In the center of this white area, the text 'Why Textual?' is written in a bold, black, sans-serif font.



... or: why not graphical?

**Languages and Editors
are easier to build**

Languages and Editors are easier to build

**Evolve Language and simple editor
as you understand and discuss the
architecture, in real time!**

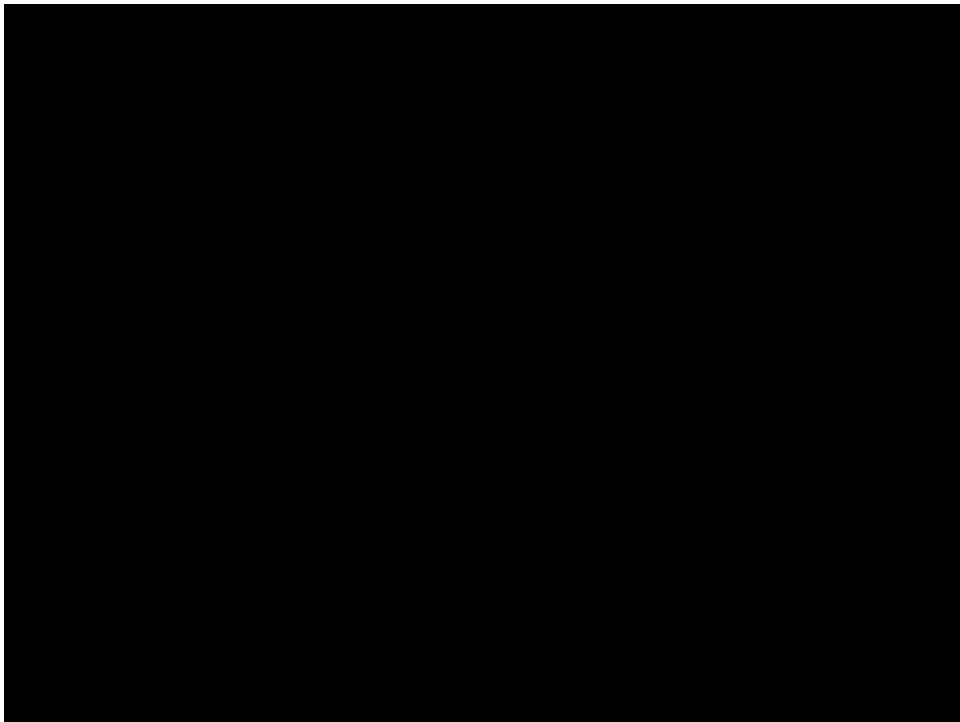
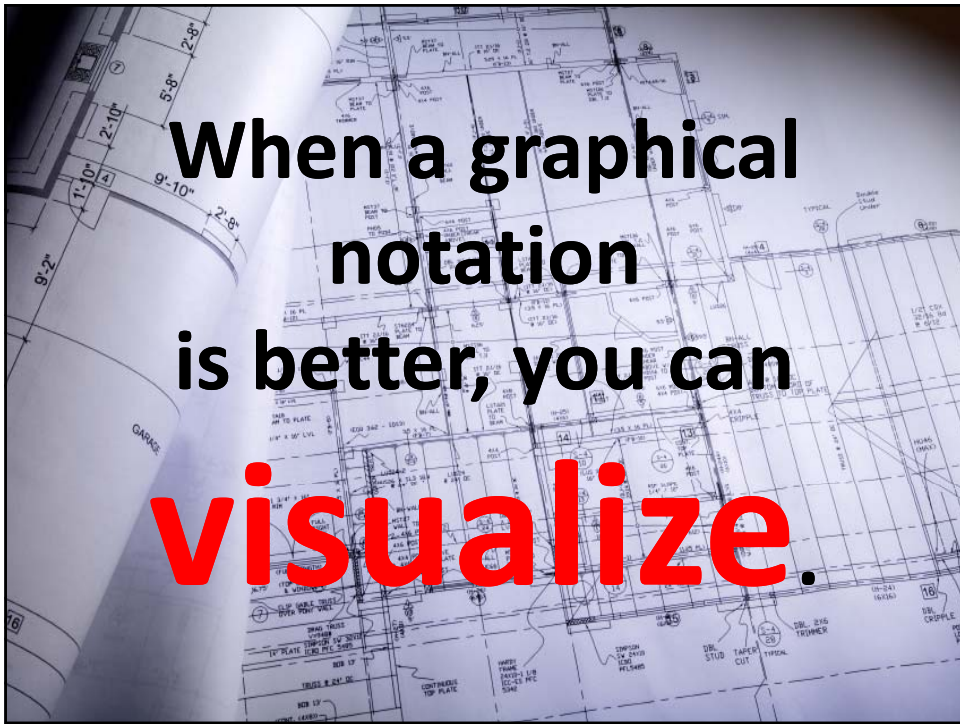
**Integrates easily with
current infrastructure:
CVS/SVN diff/merge**

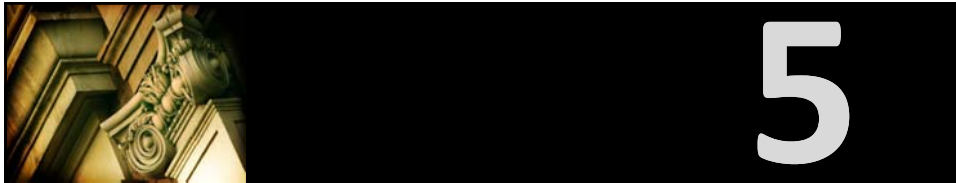
adapting existing
models as the DSL
evolves

**Model evolution
is **trivial**, you can
always use *grep*.**

**Many Developers
prefer textual
notations**








5

Tooling

**Several tools available.
Example: oAW Xtext**



Specify Grammar

```

Namespace:
  "namespace" name=ID (featureclause=FeatureClause)? "("
  (usings+-Using) *
  ( subNamespaces+=Namespace |
    components+=Component |
    datatypes+=DataType |
    interfaces+=Interface |
    compositions+=Composition ) *
  ")";

Using:
  "using" namespace=[Namespace|qualID];

Component:
  (pointcut=Pointcut)? "component" name=ID (tags=TagsClause)? (featurec
  (ports+=Port) *
  ")";

Port:
  MessagePort | DataPort;

MessagePort:
  ProvidedPort | RequiredPort;

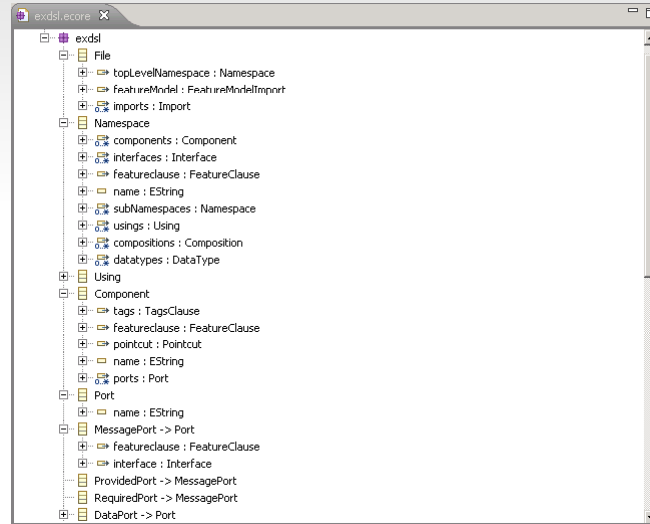
ProvidedPort:
  "provides" name=ID ":" interface=[Interface] (featureclause=FeatureCl

RequiredPort:

```

**Antlr Grammar and
Parser is generated
from this specification**

Generated Metamodel



Specify Constraints

```

import exdsl;

extension net::ample::adsl::exdsl::Extensions;
extension org::openarchitectureware::util::stdlib::io;

context Component ERROR "Qualified Name "+qualifiedName()+" must be unique"
allComponents().select( c | c.qualifiedName() == qualifiedName() ).size > 1;

context DataType ERROR "Qualified Name "+qualifiedName()+" must be unique"
allDataTypes().select( c | c.qualifiedName() == qualifiedName() ).size > 1;

context Namespace if !isEmpty() ERROR "Qualified Name "+qualifiedName()+" must be unique"
allNamespaces().select( c | c.qualifiedName() == qualifiedName() ).size > 1;

context emf::EObject if metaType.getProperty("name") != null ERROR "name not defined"
metaType.getProperty("name").get(this) != "Unnamed";

context Interface ERROR "interface names must start with a capital I":
name.startsWith("I");

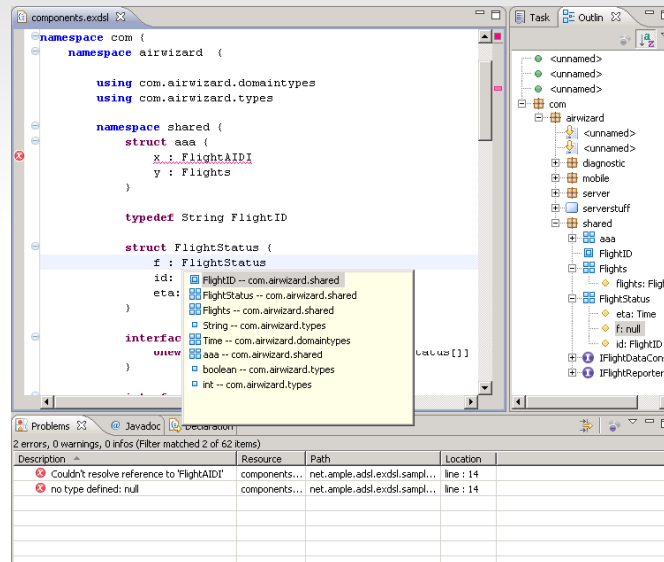
context MessagePort ERROR "interface not defined. Missing a 'using'?":
visibleInstancesOfType(this, Interface).contains(interface);

context Attribute ERROR "no type defined: "+type.name:
visibleInstancesOfType(this, DataType).contains(type);

context DataPort ERROR "data not defined: "+type.name:
visibleInstancesOfType(this, ComplexType).contains(type);

```

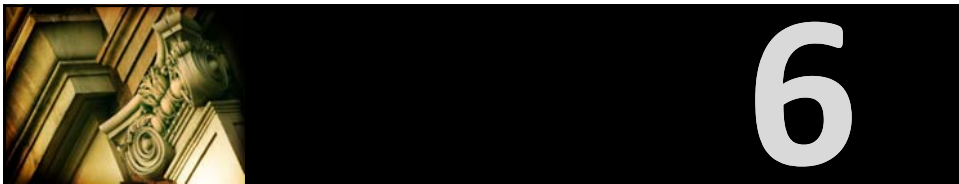
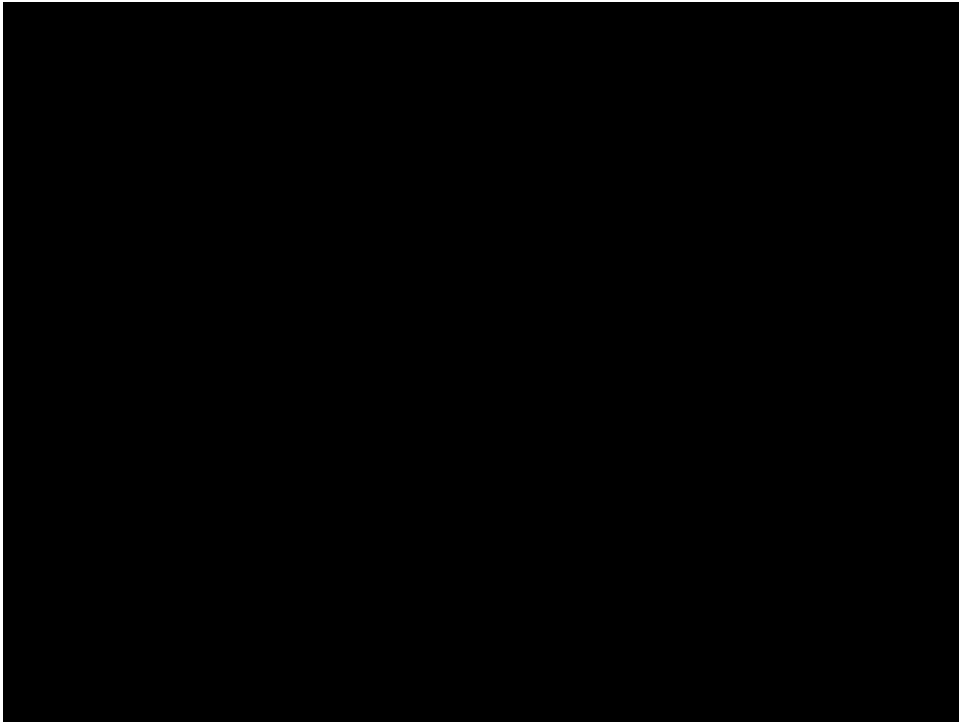
Generated Editor



DEMO II



The language-aware editor for
our DSL

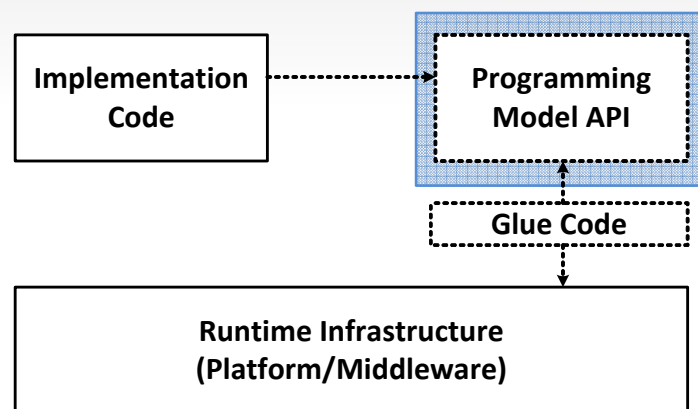


Generating Code

**Since we already
have a formal model...**

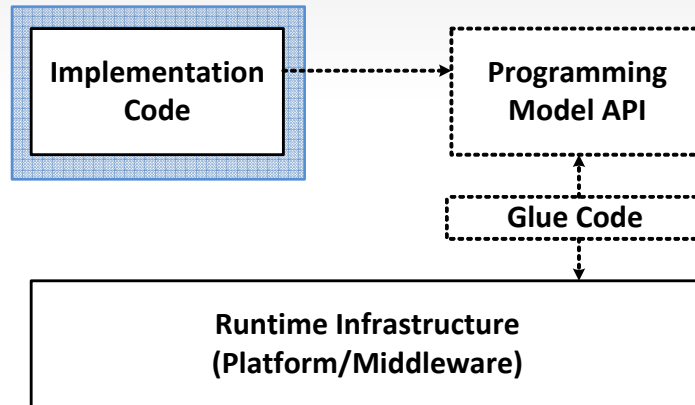
Generate API

**Maps Architectural Concepts to
Implementation language (non-trivial!)**



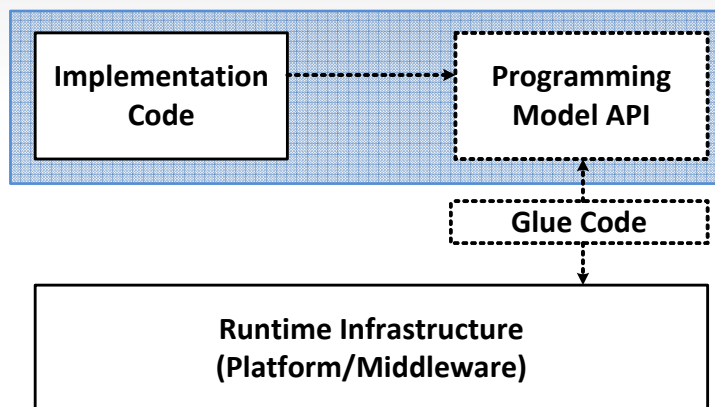
Implementation

Implementation only depends on
the generated programming model API



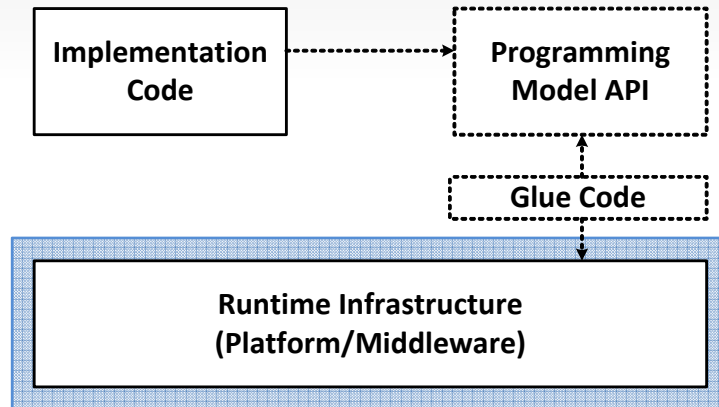
Programming Model

Generated API + Usage Idioms
Completely Technology-Independent



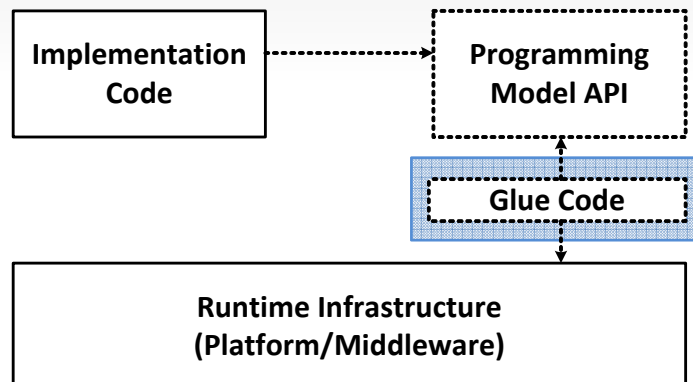
Runtime Infrastructure

Select based on fit wrt. to architectural concepts and non-functional requirements



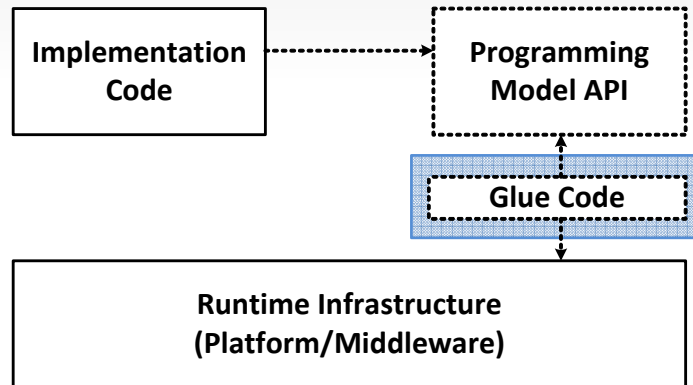
Glue Code

Aka Technology Mapping Code
Maps API to selected platform



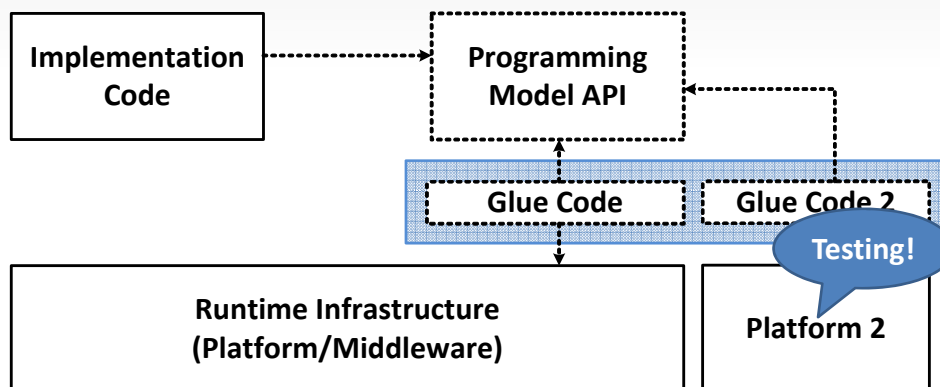
Glue Code

Contains Configuration Files for Platform
Might require „mix in models“



Several Platforms

Different Platforms, not Languages
Support for Scaling (non-functional req)



Benefits:

More Efficient Impl.
Technology Independent
Consistence/Quality
Architecture-Conformance

Code Gen Sequence

- 1) Generate API
- 2) Write Impl Code
- 3) Select Platform
- 4) Generate Glue Code

Programming
Model API

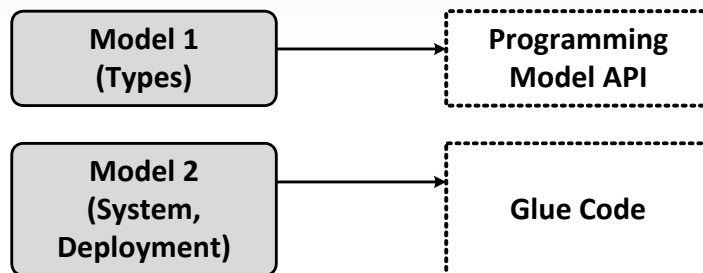
Implementation
Code

Platform

Glue Code

Separate Models

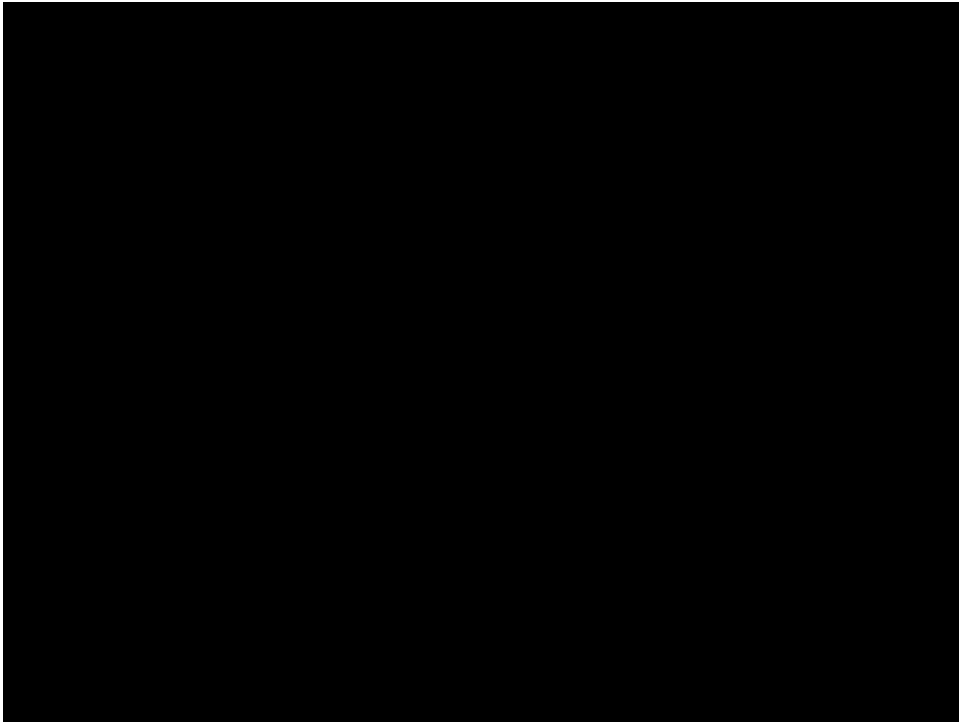
for stuff relevant for the API
vs. system/deployment stuff



DEMO III



Generating C for the
target device



7

Expressing Variability

A decorative horizontal banner at the top of the lower section features a close-up of classical architectural columns on the left and a large, white, stylized number '7' on a black background on the right. Below this banner, the text 'Expressing Variability' is centered in a bold, black, sans-serif font within a white rectangular frame.

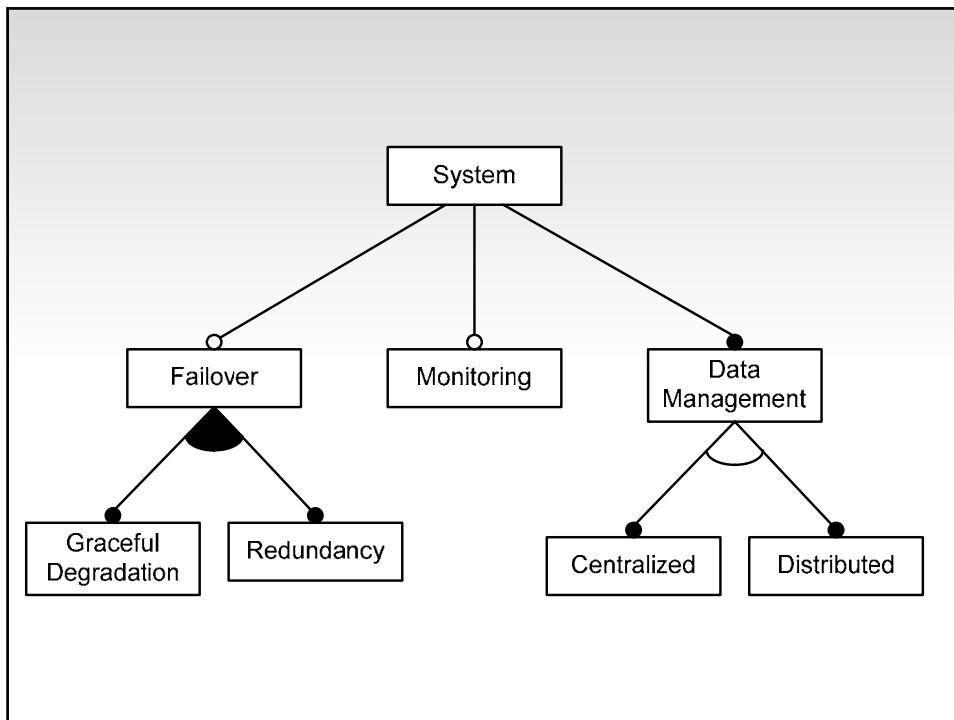
**Different Variants
of the System
for different
customers.**

**How do I
express
this in the
models?**

Negative Variability:
Conditionally taking
something away

Negative Variability:
Conditionally taking
something away

Feature Models



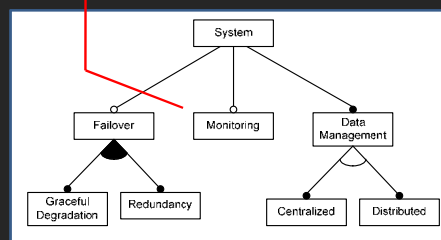
```

component DelayCalculator {
  provides default: IDelayCalculator
  requires screens[0..n]: IInfoScreen
  provides mon: IMonitoring feature monitoring
}
  
```

```

component DelayCalculator {
  provides default: IDelayCalculator
  requires screens[0..n]: IInfoScreen
  provides mon: IMonitoring feature monitoring
}

```



```

namespace monitoringStuff feature monitoring {

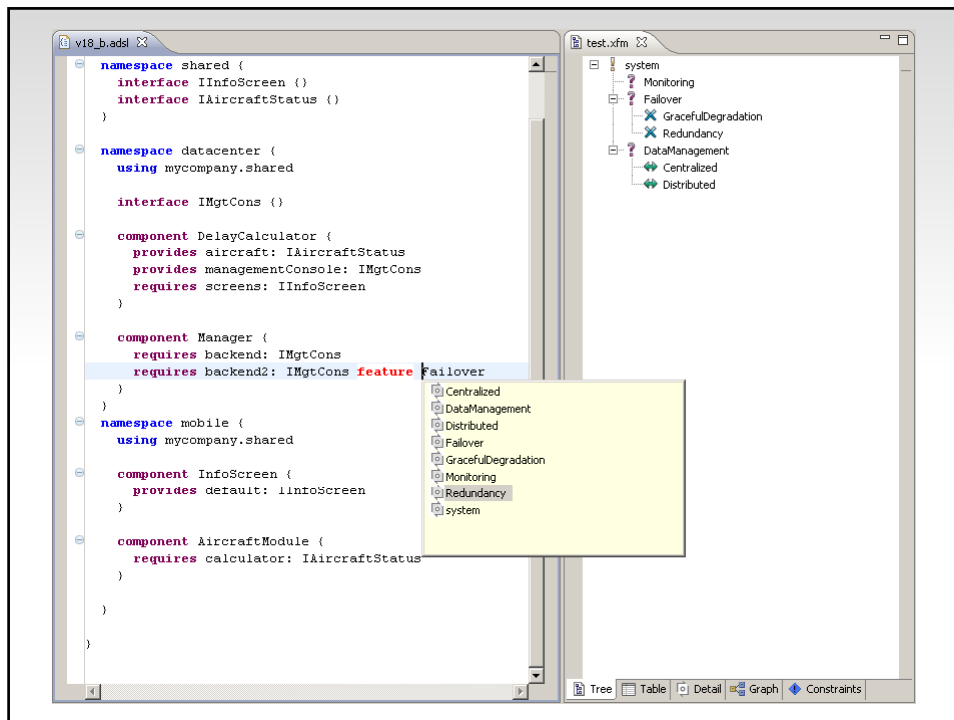
  component MonitoringConsole {
    requires devices[*]: IMonitor
  }

  instance monitor: MonitoringConsole

  dynamic connect monitor.devices query {
    type = IMonitor
  }

}

```



Positive Variability:
 Conditionally adding
 something to a
 minimal core

Positive Variability:
Conditionally adding
something to a
minimal core

Aspects

```
namespace monitoring {  
  
    component MonitoringConsole ...  
    instance monitor: ...  
    dynamic connect monitor.devices ...  
  
    aspect (*) component {  
        provides mon: IMonitoring  
    }  
}
```

```

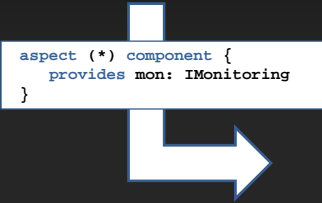
component DelayCalculator {
  ...
}
component AircraftModule {
  ...
}
component InfoScreen {
  ...
}

```

```

component DelayCalculator {
  ...
}
component AircraftModule {
  ...
}
component InfoScreen {
  ...
}

```



```

component DelayCalculator {
  ...
  provides mon: IMonitoring
}
component AircraftModule {
  ...
  provides mon: IMonitoring
}
component InfoScreen {
  ...
  provides mon: IMonitoring
}

```

Weaver is **generic**:
works with all (container)
model elements

aspect (*) <type>

all instances of *type*

aspect (tag=bla) <type>

all instances with tag bla

aspect (name=S*) <type>

all instances whose name
starts with S

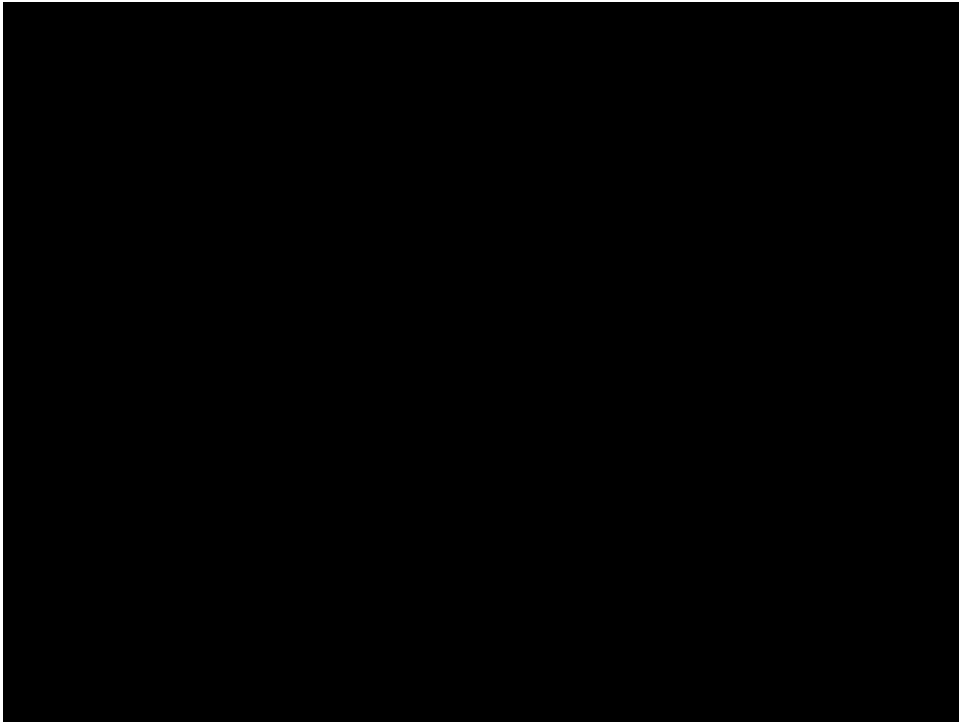
AO + Features

```
namespace monitoring feature monitoring {  
  
  component MonitoringConsole ...  
  instance monitor: ...  
  dynamic connect monitor.devices ...  
  
  aspect (*) component {  
    provides mon: IMonitoring  
  }  
}
```

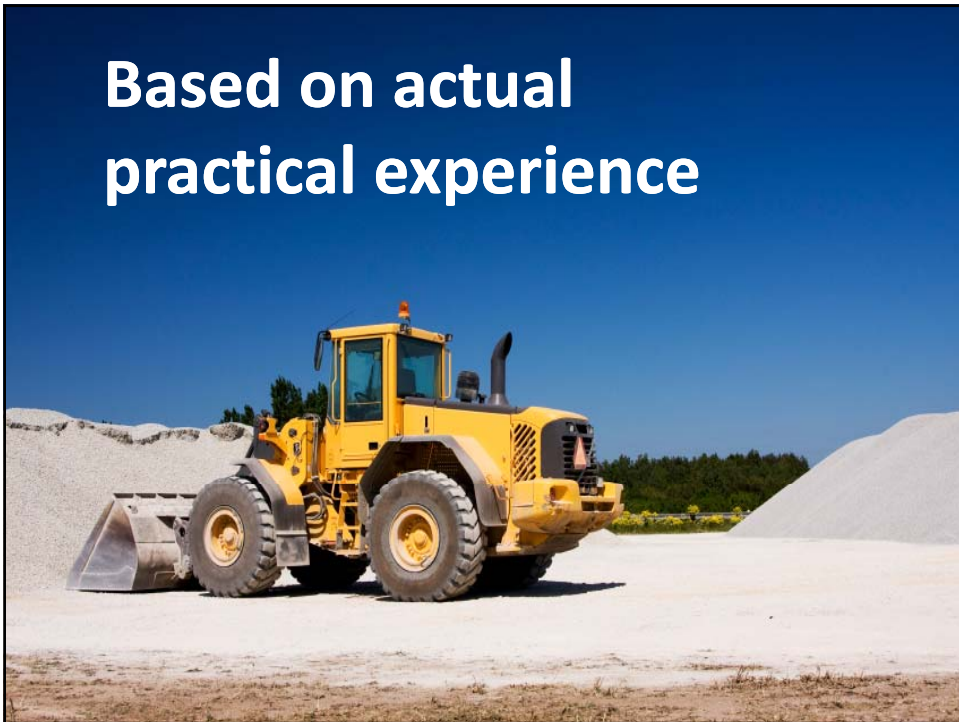
DEMO III



Adding Variability and
connectivity to a feature model
to the previous DSL



**Based on actual
practical experience**

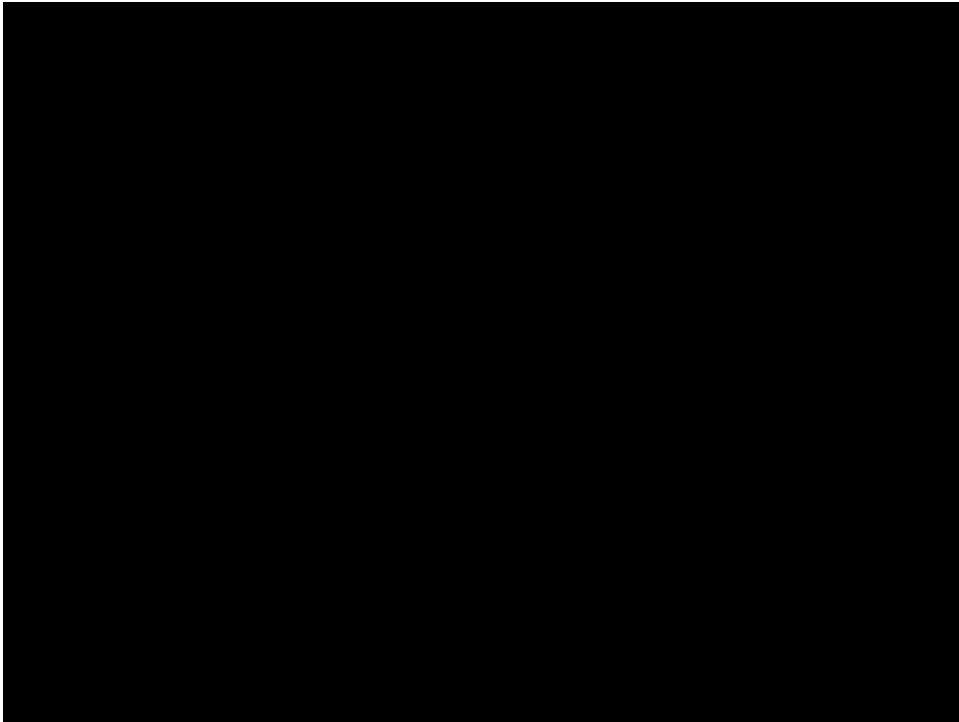


**Currently in use with
four of my customers**



**Benchmarked by
suitability for use in
today's projects**





A composite image. The top-left corner shows a close-up of classical architectural details, including a column and a decorative capital. The rest of the image is split into two main sections. On the left, there is a white rectangular area containing text. On the right, there is a photograph of a King penguin standing on a rocky beach, looking directly at the camera. Other penguins are visible in the background under an overcast sky.

THE END.
Thank you.
Questions?

