Towards improving Software Security using Language Engineering and mbeddr

Markus Völter, Zaur Molotnikov, Bernd Kolb

voelter ingenieurbüro für softwaretechnologie // itemis fortiss

voelter@acm.org
www.voelter.de
@markusvoelter

1

Software Security

Software security refers to the security properties of a software system's implementation.

G. McGraw,
Software Security: Building Security In.
Addison-Wesley, 2006.

Software security

- Techniques
  - Programming Language
    (and the wrong use of it)
- Process
  - Education
  - Awareness
  - Reviews

C

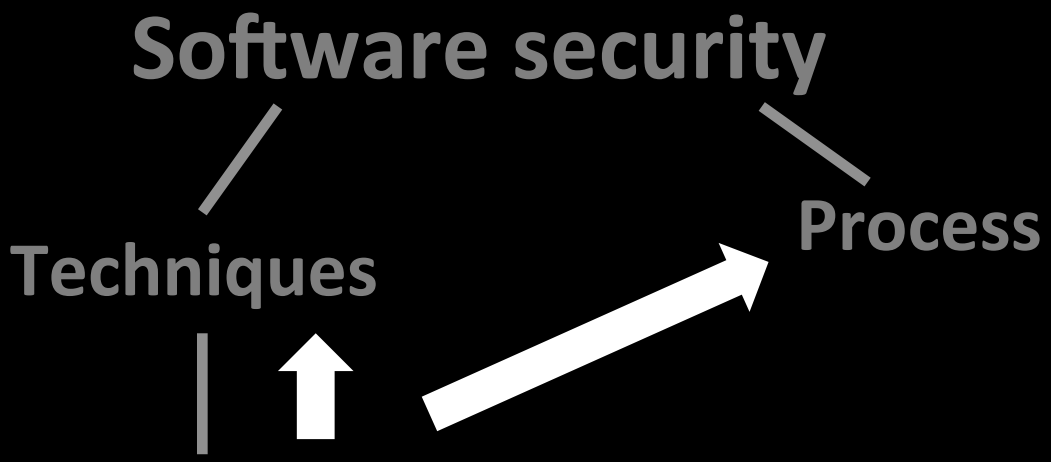
Software security

Techniques

Process

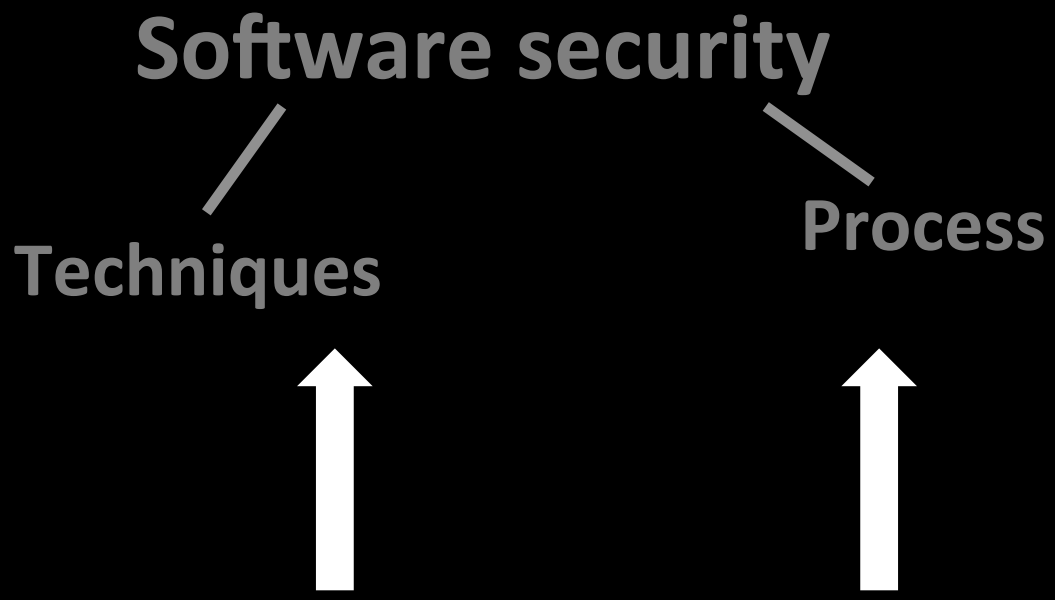
Programming Language

C



C

Make C better and less dangerous through suitable language extensions.



C

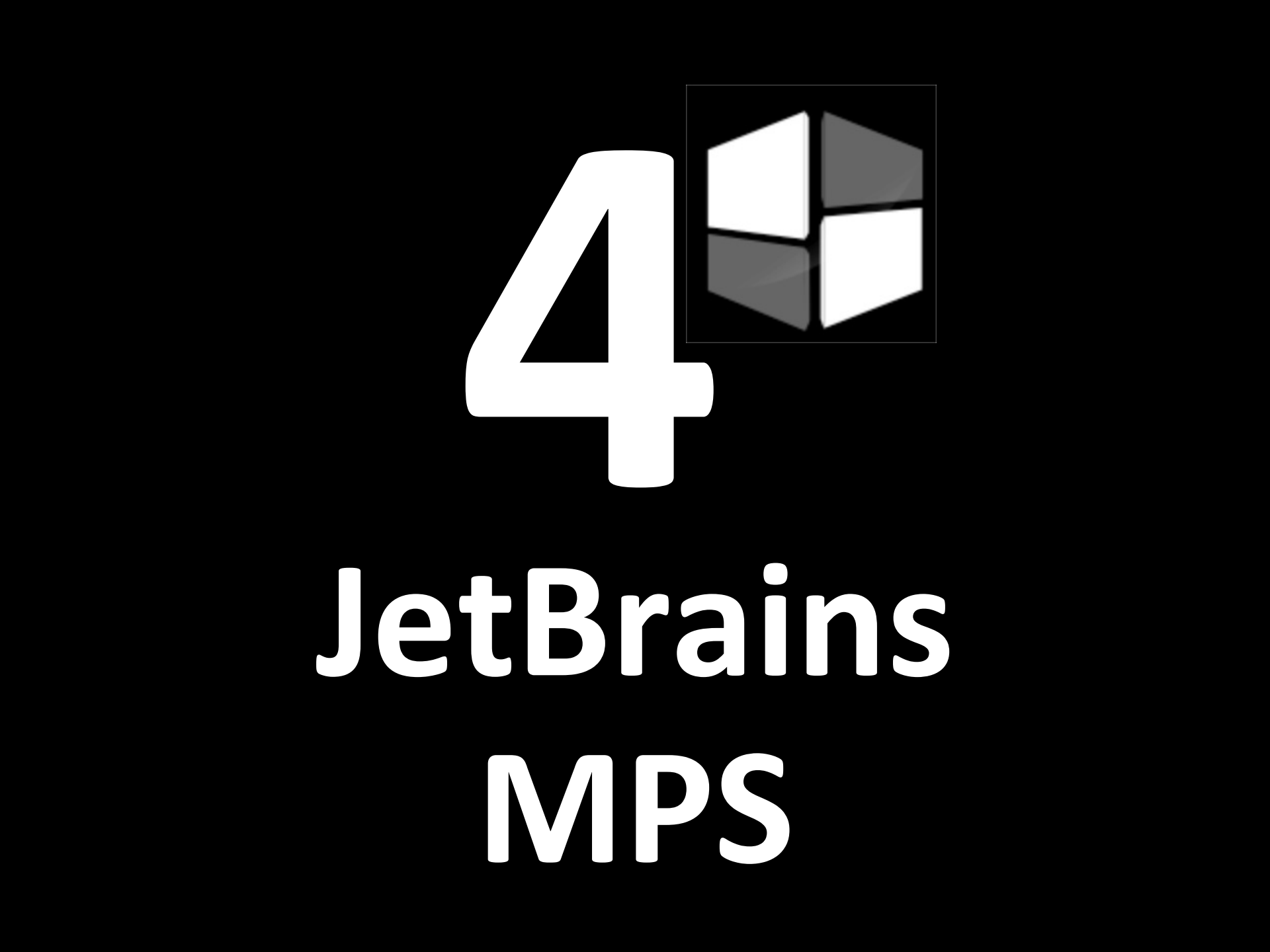
Make C better and less dangerous through suitable language extensions.

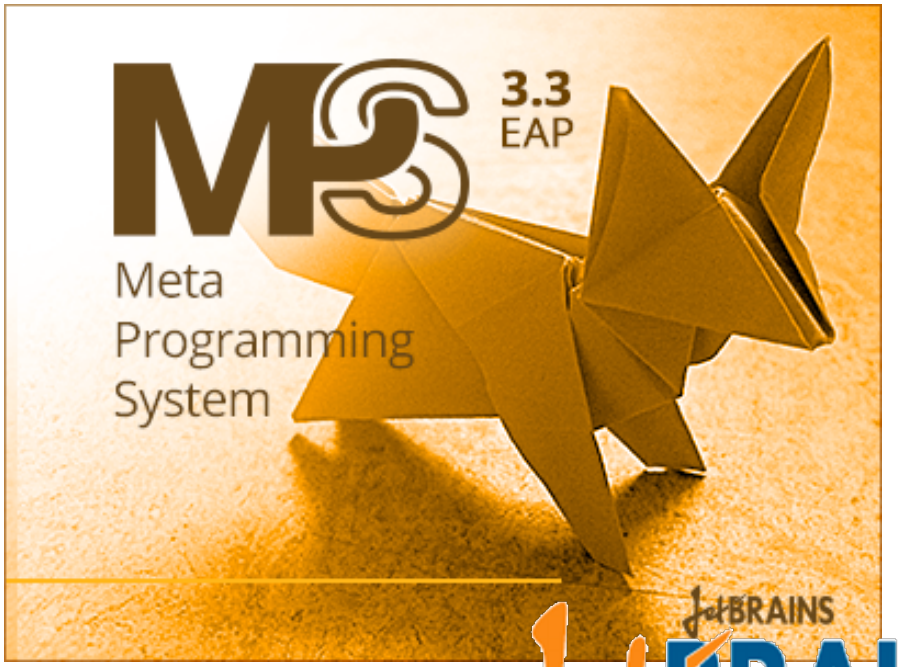
2

Language Engineering

3

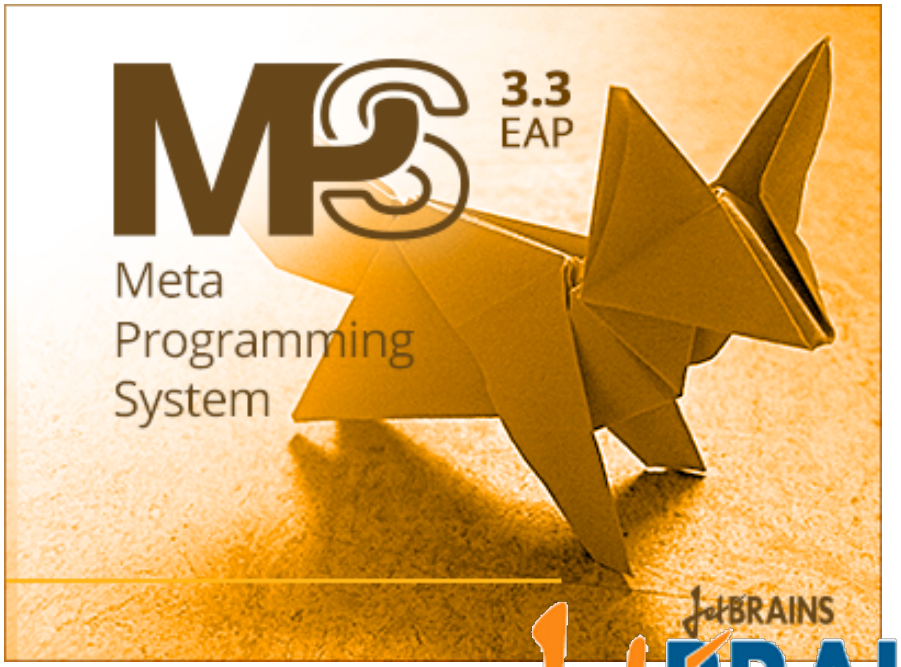
Language Workbenches


4
JetBrains
MPS



A Language Workbench –

a tool for defining, composing and using ecosystems of languages.



Open Source

Apache 2.0

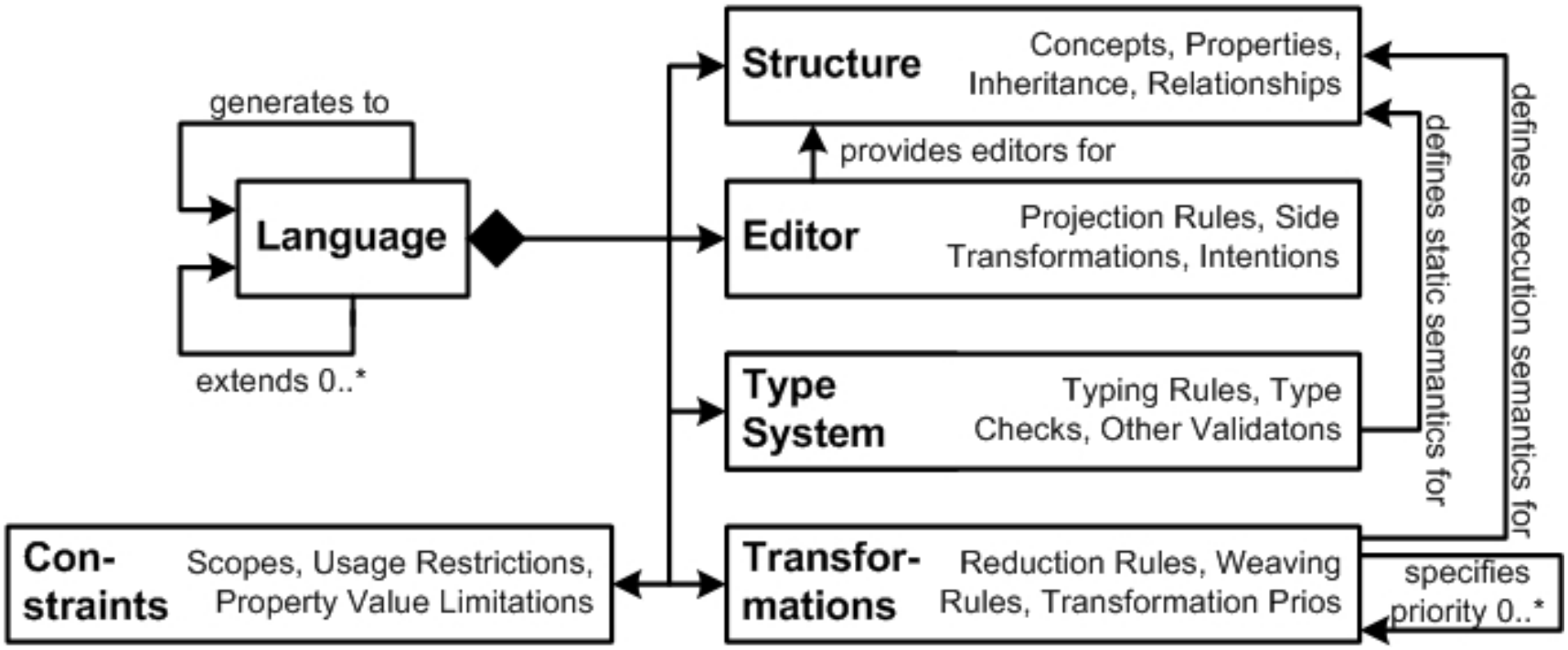
http://jetbrains.com/mps

V 3.2 is current.

V 3.3 in RC – Release Q4 2015.

[Language Workbench]

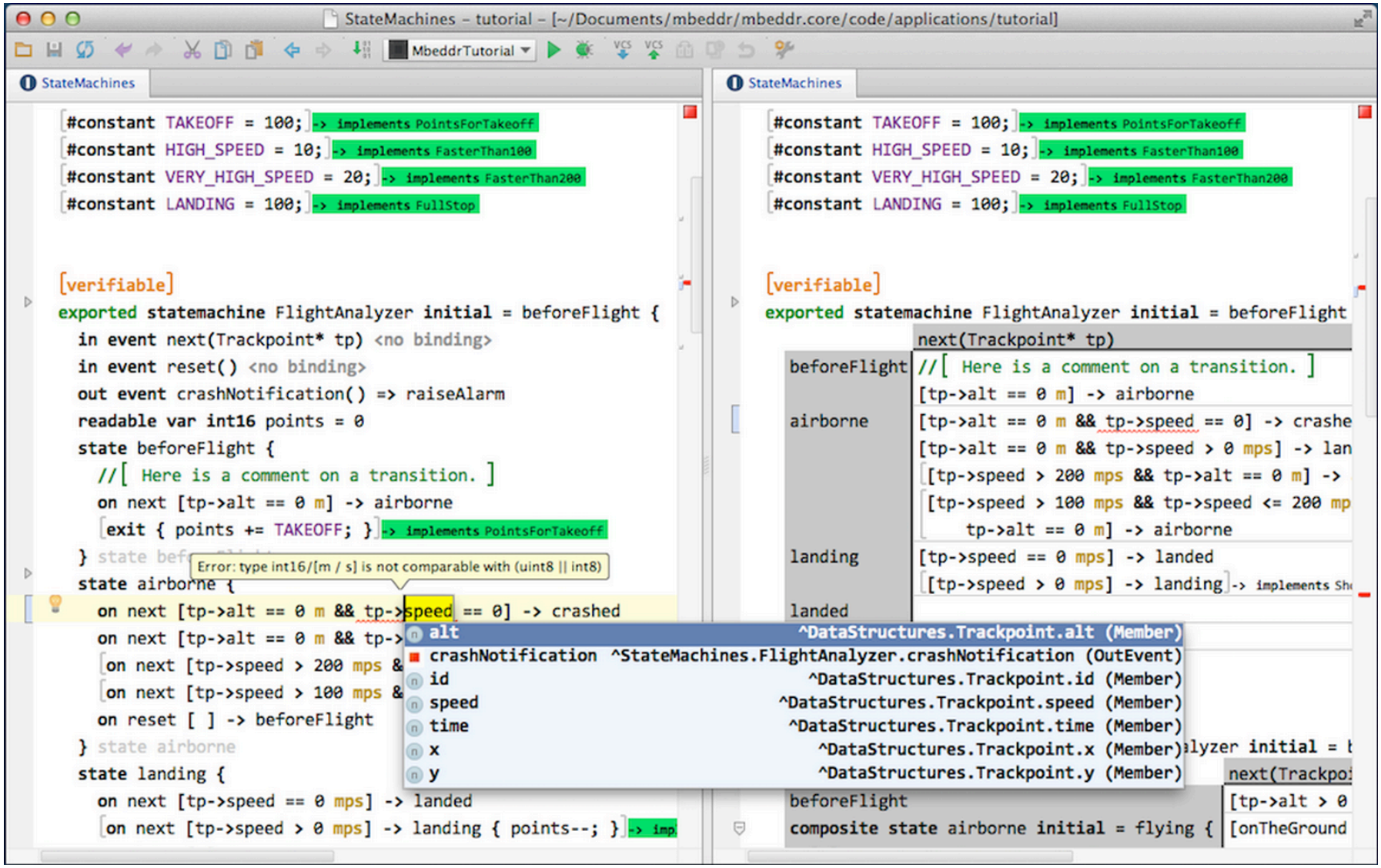
Comprehensive Support for many aspects of Language Definition.



+ Refactorings, Find Usages, Syntax Coloring, Debugging, ...

[Comprehensive IDE Features]

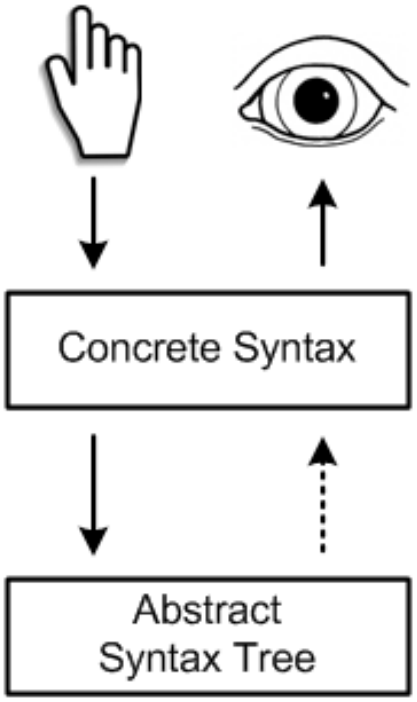
For End Users and Language Developers



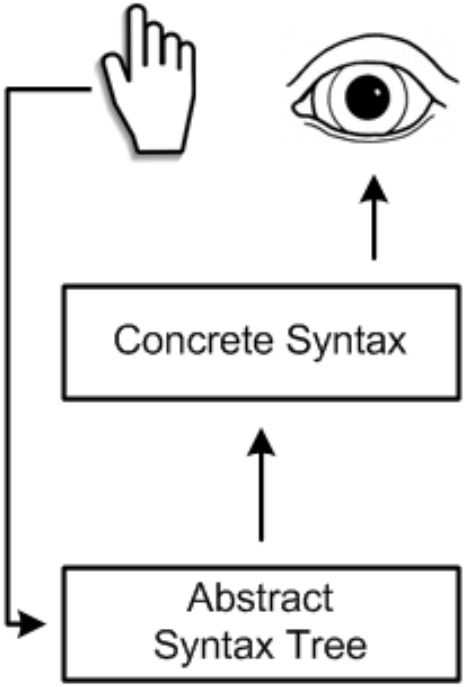
[Projectional Editing]

MPS uses a Projectional Editor

Traditional: Parsing



MPS: Projectional Editing

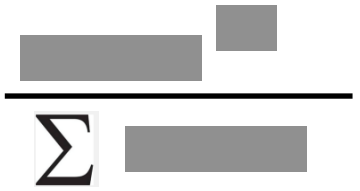


A Projectional Editor modifies the AST directly.
No grammars or parsers are involved.

[Projectional Editing]
Advantage: Syntactic Flexibility

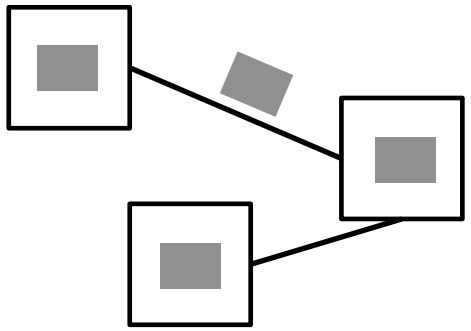
Regular Code/Text

Mathematical



Tables

Graphical



[Projectional Editing]
Advantage: Syntactic Flexibility / MPS

Regular Code/Text

```
//[ A documentation comment with references
  [ to @arg(data) and @arg(dataLen)
void aSummingFunction(int8[ ] data, int8 dataLen) {
  int16 sum;
  for (int8 i = 0; i < dataLen; i++) {
    sum += data[i];
  } for
} aSummingFunction (function)
```

Mathematical

```
double midnight2(int32 a, int32 b, int32 c) {
  return (-b + √(b^2 - Σ_{i=1}^{4} a * c)) / (2 * a);
} midnight2 (function)
```

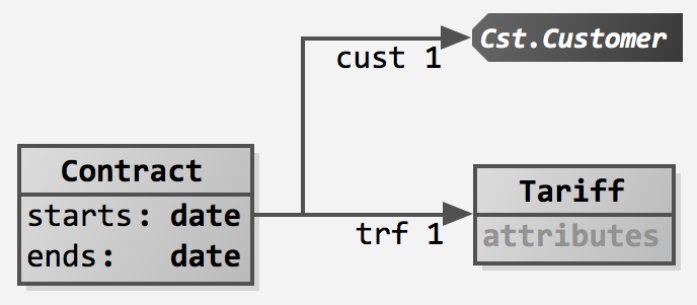
Tables

```
int16 decide(int8 spd, int8 alt) {
  return
```

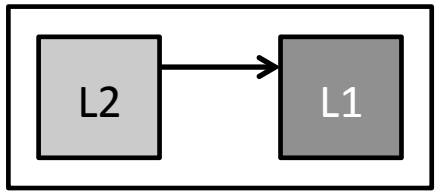
	spd > 0	spd > 100
alt < 0	1	1
alt == 0	10	20
alt > 0	30	40
alt > 100	50	60

```
otherwise 0;
} decide (function)
```

Graphical



[Projectional Editing]
Advantage: Language Composition



Separate Files

Type System
Transformation
Constraints

In One File

Type System
Transformation
Constraints
Syntax
IDE

mbeddr

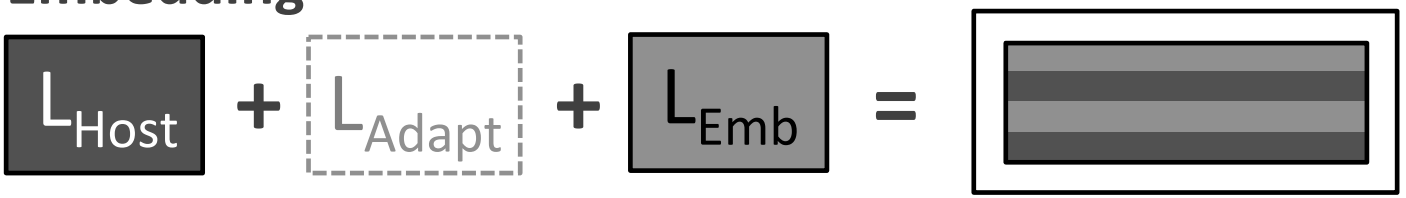
5+ base languages
50+ extensions to C
10+ extensions to requirements lang.

[Projectional Editing]

Advantage: Language Composition

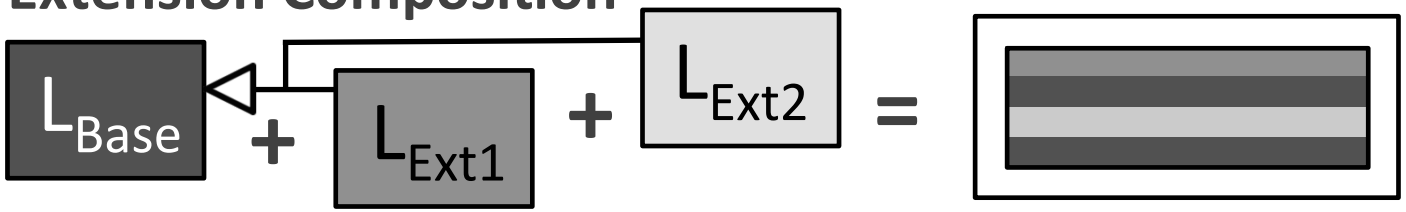
No change to definition of L1 or L2 in order to use them together.

Embedding



Extension

Extension Composition



[Projectional Editing]
Study Results on Editor Usability

People **prefer MPS** over conventional IDEs

MPS more is **more efficient** than normal IDEs

MPS more is **more productive** than normal IDEs

MPS makes it easier to create **correct programs**

MPS enforces a **structurally correct AST**

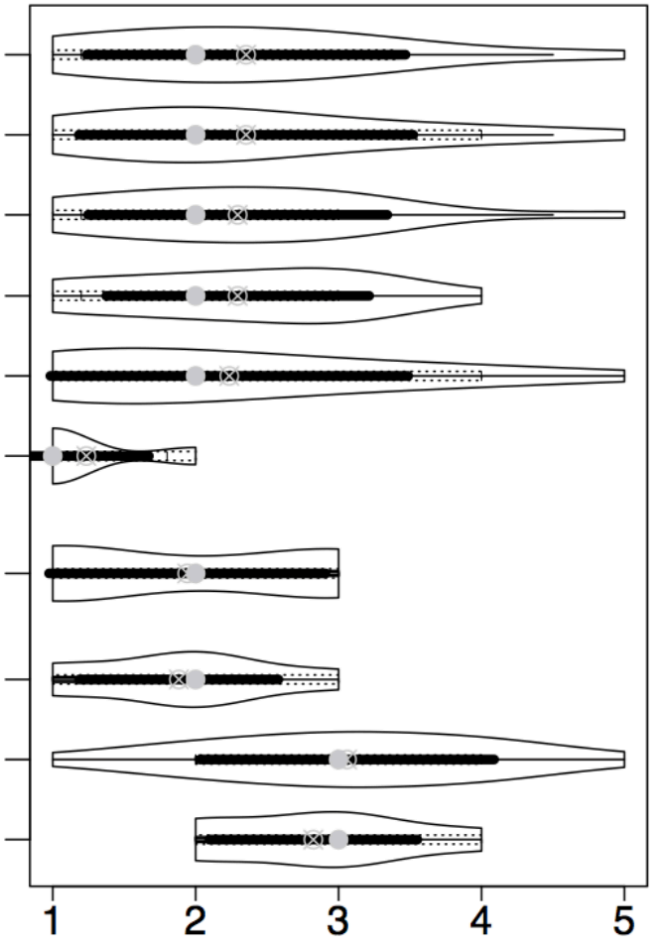
People benefit from **language modularity**

People benefit from the **flexible notations**

People benefit from **advanced navigation support**

The experience with **learning MPS** is mixed.

It takes some time to **get used to MPS**



Further Reading

Projecting a Modular Future
Three different case studies of using MPS
http://voelter.de/data/pub/projectingModuleFuture.pdf

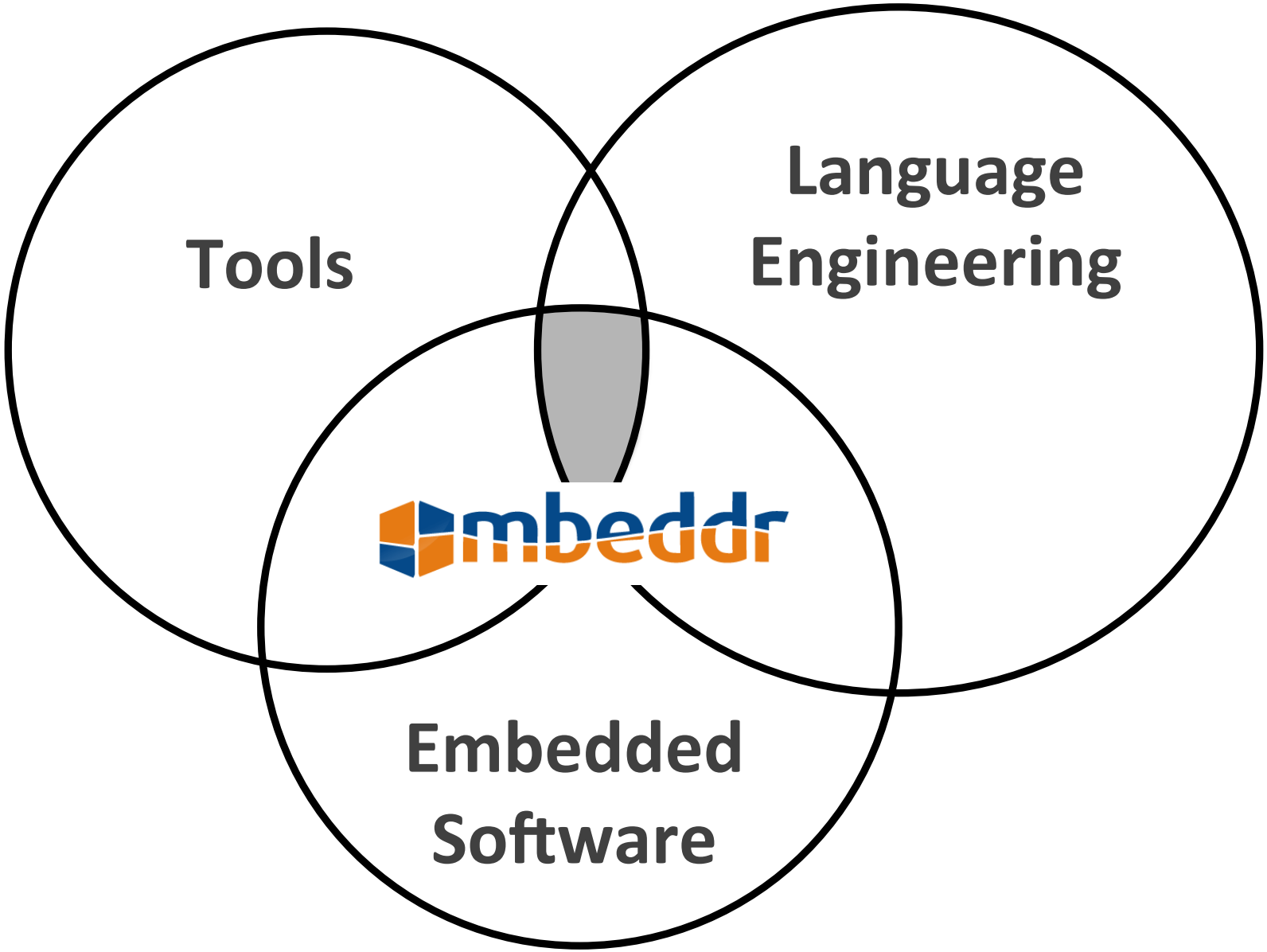
mbeddr -- Instantiating a Language Workbench in the Embedded Software Domain
Detailed Discussion of mbeddr (using MPS for embedded s/w engineering)
http://voelter.de/data/pub/voelteretal-mbeddr-AUSE.pdf

Towards User-Friendly Projectional Editors
Study about the usability of projectional editors and MPS' „tricks" for improving it
http://voelter.de/data/pub/projectionalEditing-sle2014.pdf

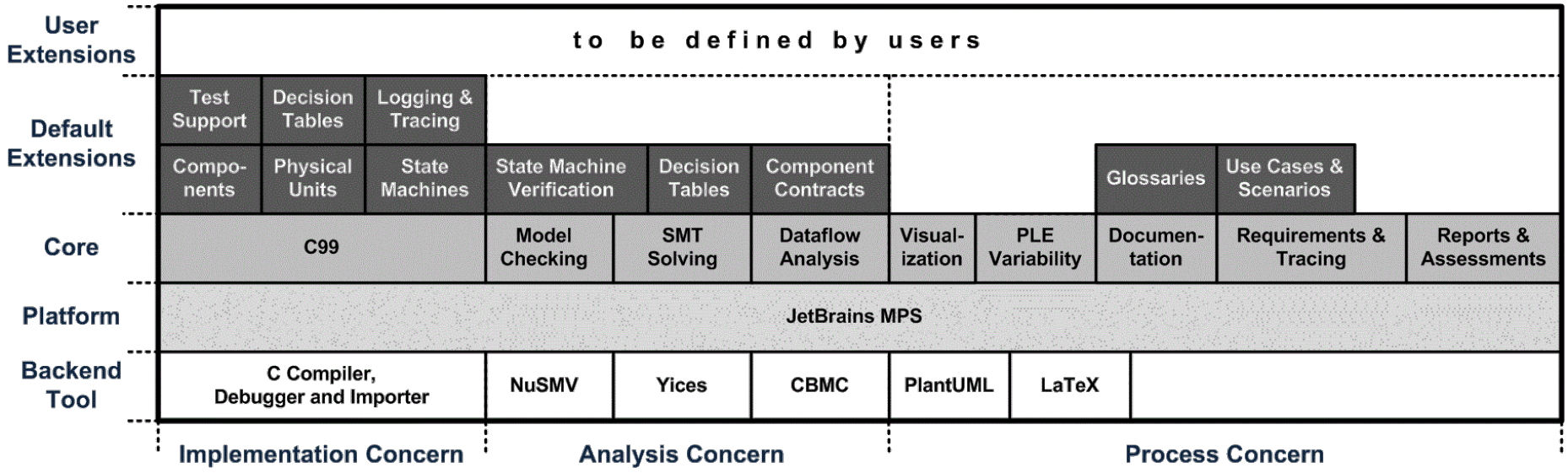
The State of the Art in Language Workbenches – LWC Conclusions
Systematic Comparison of different language workbenches and their features
http://voelter.de/data/pub/LWC13.pdf

5

mbeddr


Tools
Language
Engineering
mbeddr
Embedded
Software

An extensible set of integrated languages for embedded software engineering.



itemis

fortiss

BMW Car IT

Open Source @ eclipse.org
Eclipse Public License 1.0
http://mbeddr.com

Bundesministerium
für Bildung
und Forschung

7 developers, project management

2 developers, verification support

1 developer, verification support

3 developers, C++

---

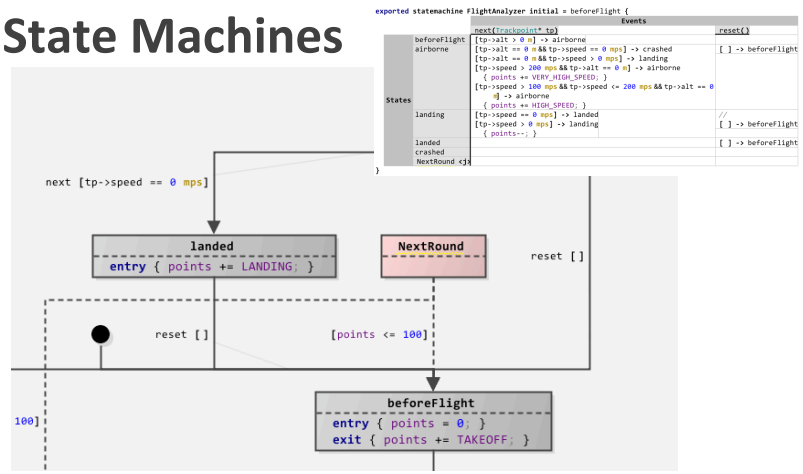
strategic collaboration with

Some of the C Extensions

Units

```
void calcVerticalSpeed(TWP* prev, TWP* cur) {
  if (prev == null) {
    cur->vSpeed = 0 mps;
  } else {
    int16/m/ dAlt = cur->alt - prev->alt;
    int8/s/ dTime = cur->time - prev->time;
    cur->vSpeed = dTime / dAlt;
  }
} calcVerticalSpeed (function)
```

State Machines



Components

```
exported cs interface TrackpointProcessor {
  Trackpoint* process(Trackpoint* p)
    pre(0) p != null
    pre(1) p->id != 0
    pre(2) p->time != 0 s
    post(3) result->id != 0
}

exported component Nuller extends nothing {
  provides TrackpointProcessor processor
  Trackpoint* processor_process(Trackpoint* p) <= op processor.process {
    p->alt = 42 m;
    return p;
  } runnable processor_process
} component Nuller
```

Math

```
int32 averageIntArray(int32[ ] arr, int32 size) {
  return (sum i = 0 to size arr[i]) / size;
} averageIntArray (function)

double midnight1(int32 a, int32 b, int32 c) {
  return (-b + sqrt(b^2 - 4 * a * c)) / (2 * a);
} midnight1 (function)
```

An IDE for Requirements

Requirements

6.1 | You should land as short as possible
`ShortLandingRoll /functional: tags`

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Suspendisse potenti. Etiam risus ante, bibendum ut mattis
vel condimentum velit. Quisque venenatis faucibus tellus
Phasellus rhoncus quam eu dui dictum sollicitudi

6.2 | Once you land successfully, you get another
`FullStop /functional: tags`

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Suspendisse potenti. Etiam risus ante, bibendum ut mattis
purposes, this one references **@req**(InFlightPoints)

Rules

calculation PointForATrackpoint: This rule computes the points
It does so by taking into accou
passed as arguments.

parameters: [**int16** alt: current altitude of the trackpoint
int16 speed: current speed of the trackpoint]

result = (*BASEPOINTS* * 1) *

	alt > 2000	alt > 1000
speed > 180	30	15
speed > 130	10	20

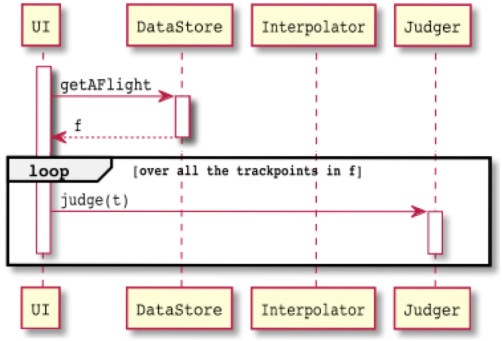
tests:
```
PointForATrackpoint(500, 100) == 0
PointForATrackpoint(500, 1200) == 0
PointForATrackpoint(1100, 165) == 200
PointForATrackpoint(2100, 140) == 100
PointForATrackpoint(2100, 200) == 300
```

Tracing

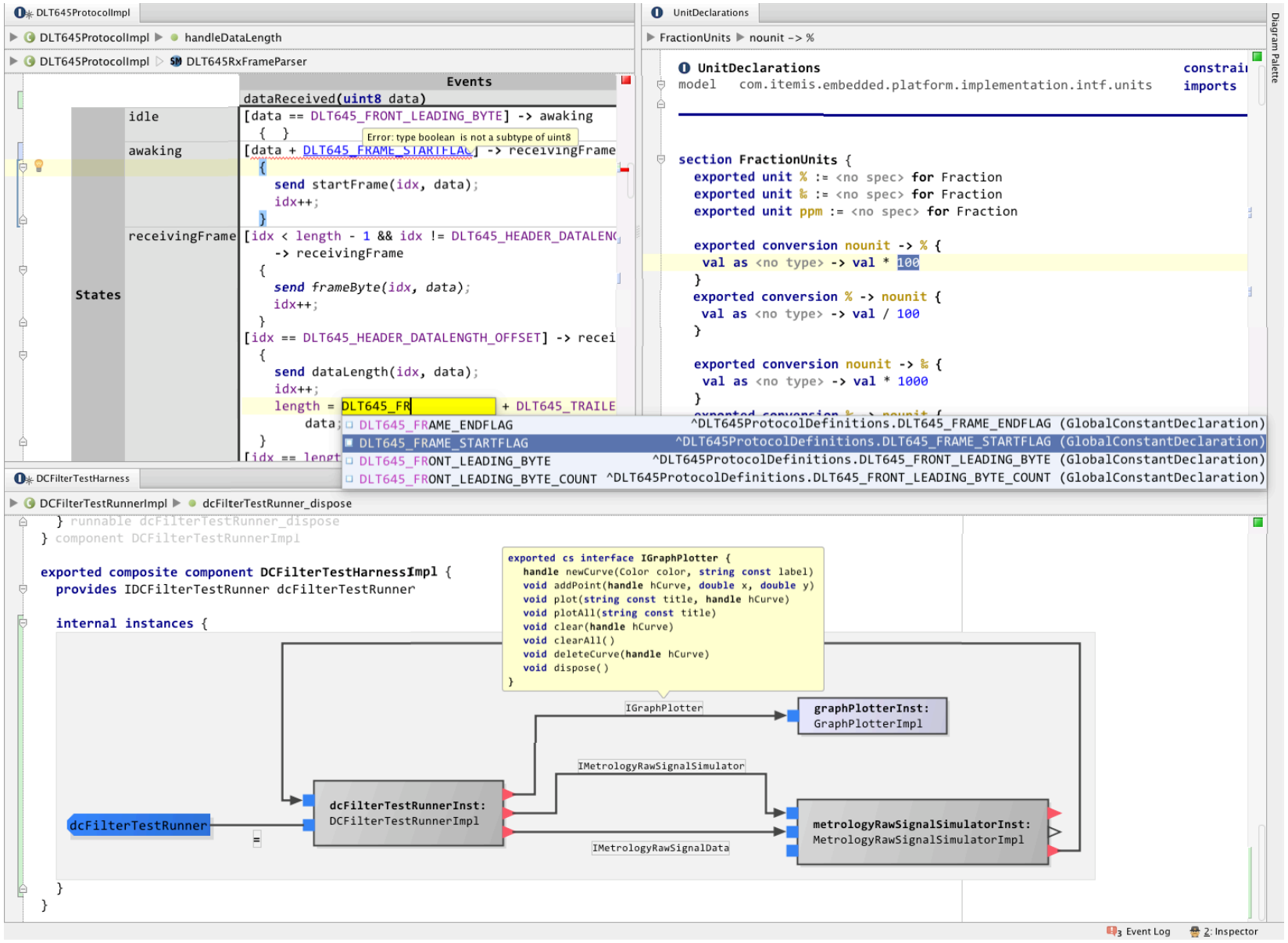
```
#constant TAKEOFF = 100; -> implements PointsForTakeoff
#constant HIGH_SPEED = 10; -> implements FasterThan100
#constant VERY_HIGH_SPEED = 20; -> implements FasterThan200
#constant LANDING = 100; -> implements FullStop

[checked]
exported statemachine FlightAnalyzer initial = beforeFli
  state beforeFlight {
    entry { points = 0; }
    on next [tp->alt > 0 m] -> airborne
    [exit { points += TAKEOFF; } -> implements PointsForTakeoff
  } state beforeFlight
```

Visualisations



Great IDE Support



mbeddr

An IDE for Documentation

5.4 Stripping and Reintroducing Units

Let us assume we have an existing (legacy or external) function that does not know about physial units and you cannot or do not want to use generic units. An example is anExistingFunction:

```
int16 anExistingFunction(int16 x) {
  return x + 10;
} anExistingFunction (function)
```

Code from anExistingFunction

To be able to call this function with arguments that have units, we have to strip away the units before we call the function. This can be achieved by selecting the corresponding expression and invoking the **Strip Unit** intention. The type of this stripped expression will be simply the type of the original expression but without units.

```
int16/m/ someFunction(Trackpoint* p1, Trackpoint* p2) {
  int16 newValueWithoutUnit = anExistingFunction(stripunit(p1.alt));
  return newValueWithoutUnit m;
} someFunction (function)
```

6

Solution: Technique

Code Markup and Checking

```
int16/m/ dAlt = cur->alt - prev->alt;
int8/s/ dTime = cur->time - prev->time;
cur->vSpeed = dTime / dAlt;
```

Error: type int16 /s · m^(−1)/ is not a subtype of int16 /mps/

Robustness ⊂ Security

Layers checking, access permissions

„Units“ for sanitized or encrypted data

Straightforward Language Extensions

Apple GotoFail bug:

```
if (validateStep1(data, ...) != 0) goto fail;
if (validateStep2(data, ...) != 0) goto fail;
if (validateStep3(data, ...) != 0) goto fail;
fail: handleFailedValidation(data, errorcode, ...);
```

More robust construct:

```
trysequentially {
  validateStep1(data, ...);
  validateStep2(data, ...);
  validateStep3(data, ...);
} on fail (errorcode) {
  handleFailedValidation(data, errorcode, ...);
}
```

Straightforward Language Extensions

Better than macros:

```
#define GUARD(x) if ((x)<0) return -1
```

Amazon's s2n library

Danger:

```
GUARD( do_something ) + 1;
```

Macros have no protection against wrong use; Poor man's LE. Real LE is more robust/safer.

Adapting Semantics

```
char* encryptData(char* k_enc, char* data) {
    char k_clr[256];
    decryptKey(k_enc, k_clr);
    char* encryptedData = // encrypt with k_clr
    return encryptedData;
}
```

When leaving the function, **k_clr** still on the stack – only stack pointer moved.

Better:

As variables leave scope, actually wipe their memory.

Exploiting the Generation Step

Timing Side Channel Attacks:

Insert random waits.

```
/* Returns 1 if a and b are equal, in constant time */
int s2n_constant_time_equals(string, string, len);
```

Amazon's s2n library

Naming conventions not checkable.

Semantic Annotations better.

Then insert **busy wait** at end enforce constant time.

Additional Constraints

Detect insecure C functions:

strcpy and the like.

Mark vetted secure functions as *SECURE API* and only use those.

Check this with tool.

Verification I

Heartbleed bug:

```
struct {
  uint16 payload_length;
  unsigned char payload[payload_length];
} HeartbeatMessage;
```

The above code is invalid in C

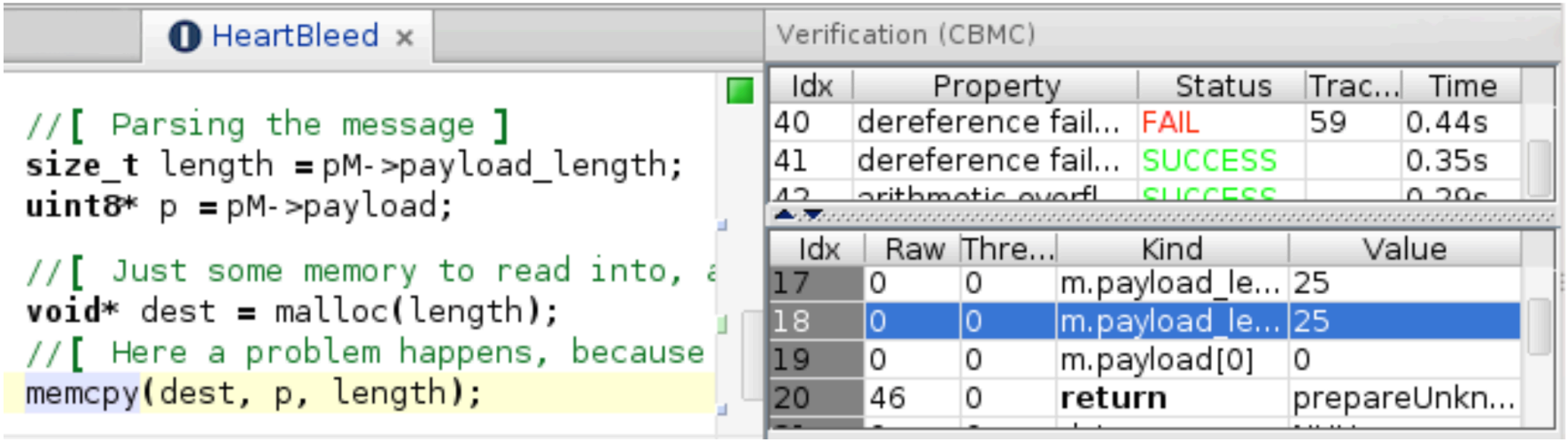
cannot „dynamically“ initialize length.

Bug: sizeof(payload) != payload_length

Verification II

Find Problem via formal Verification

```
HeartbeatMessage prepareUntrustedMessage() {
  HeartbeatMessage msg;
  assign nondet msg;
  return msg;
}
```



Verification III

Even better:

First-class message concept
that handles low level message stuff.

```
message CurrentMeasuredValue:42 {
  int32     timestamp; // time of measurement
  uint16/A/ value;     // measured value in Amps
  uint16    accuracy;  // accuracy in 1/100 %
}
message ... { ... }
...
```

```
// a field representing a times
uint8[6] f_time = {0x00A,  // f
                   UNIT_TIME24,
                   3, // 3 payl
                   10, 20, 00 /
                  };

// a field representing a measured value
uint8[4] f_value = {0x04D,  // field type identifier
                    UNIT_QDOT, // unit used: mass flow
                    1, // 1 payload byte follows
                    &dataField // addr of variable
                   };

// a message that uses the two fields
uint8[5] message = {0xAEE, // message type idenfier
                    ID, // unique running message ID
                    2, // two fields folllowing
                    f_time, // embed the time field
                    f_value// embed the value field
                   };
```

Solution: Process

Better Abstraction, Better Review

This is better than a switch-case:

```
statemachine TrafficLights initial = red {
  in event timer(int64 t) <no binding>
  in event buttonPressed() <no binding>
  var boolean button = false
  var int64 tEnter = 0
  state red {
    on buttonPressed [ ] -> red { button = true; }
    on timer [t - tEnter > 1000] -> green { tEnter = t; }
  } state red
  state green {
    on timer [t - tEnter > 500] -> red { tEnter = t; }
  } state green
}
```

```
statemachine TrafficLights initial = red {
```

		Events	
		timer(int64 t)	buttonPressed()
States	red	[t - tEnter > 1000] -> green { tEnter = t; }	[ ] -> red { button = true; }
	green	[t - tEnter > 500] -> red { tEnter = t; }	

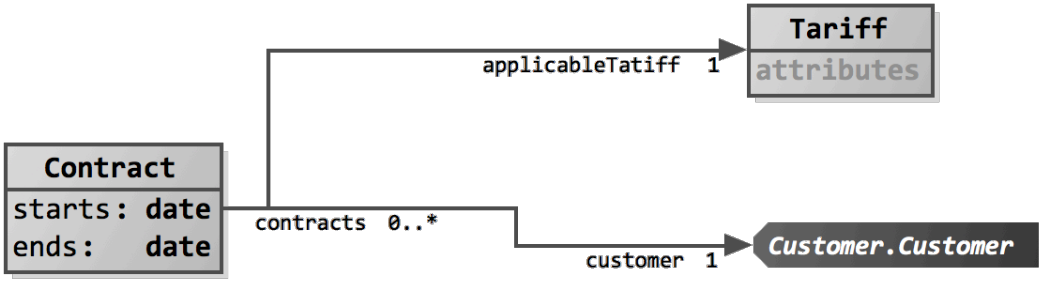
```
}
```

Better Notation, Better Review

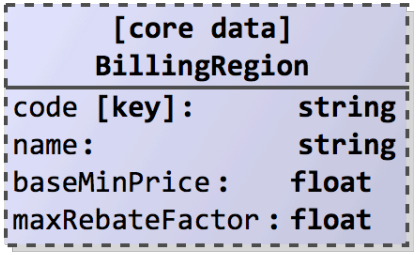
```
int32 averageIntArray(int32[] arr, int32 size) {
```

$$\text{return } \frac{\sum_{i=0}^{size} arr[i]}{size};$$

```
} averageIntArray (function)
```



Code	Name	Base Min Price	Max Rebate
BW	Baden Württemberg	0.20	0.8
BY	Bayern	0.20	0.8
BE	Berlin	0.15	0.7
BB	Brandenburg	0.10	0.7
HB	Bremen	0.20	0.7
HH	Hamburg	0.15	0.7
HE	Hessen	0.15	0.7
MV	Mecklenburg-Vorpommern	0.10	0.7
NI	Niedersachsen	0.15	0.7
NW	Nordrhein-Westfalen	0.15	0.7
RP	Rheinland-Pfalz	0.15	0.7
SL	Saarland	0.15	0.7
SN	Sachsen	0.10	0.7
ST	Sachsen-Anhalt	0.10	0.7
SH	Schleswig-Holstein	0.15	0.7
TH	Thüringen	0.10	0.7



Tracing

Trace from **every** program element to requirements (internal or external)

Analyze and find untraced program fragments.

Different kinds of traces to express semantics.

```
#constant LANDING = 100; -> implements FullStop
exported statemachine FlightAnalyzer initial = beforeFlight {
  state beforeFlight {
    on next [tp->alt > 0 m] -> airborne
    [exit { points += TAKEOFF; }] -> implements PointsForTakeoff
  } state beforeFlight
```

Expressing Security Requirements

6.1 You should land as short as possible
ShortLandingRoll /functional: tags

Lorem ipsum dolor sit amet, consectetur adipiscing elit. I
Suspendisse potenti. Etiam risus ante, bibendum ut mattis
vel condimentum velit. Quisque venenatis faucibus tellus
Phasellus rhoncus quam eu dui dictum sollicitudi

6.2 Once you land successfully, you get another
FullStop /functional: tags

Lorem ipsum dolor sit amet, consectetur adipiscing elit. I
Suspendisse potenti. Etiam risus ante, bibendum ut mattis
purposes, this one references **@req**(InFlightPoints)

calculation PointForATrackpoint: This rule computes the points a
It does so by taking into accou
passed as arguments.

parameters: [**int16** alt: current altitude of the trackpoint
int16 speed: current speed of the trackpoint] =

result = (*BASEPOINTS* * 1) *

	alt > 2000	alt > 1000
speed > 180	30	15
speed > 130	10	20

tests:
```
PointForATrackpoint(500, 100) == 0
PointForATrackpoint(500, 1200) == 0
PointForATrackpoint(1100, 165) == 200
PointForATrackpoint(2100, 140) == 100
PointForATrackpoint(2100, 200) == 300
```

Requirements are Prose, plus arbitrary formal parts, inline in the document.

Requirements can be classified as ***secure***.

Different review policy, cannot be changed (see next)

Code Review and Security Audits

Track review state in the code

```
[Review: ready   Reviewed 6 days ago by zaur ]
exported component Judge2 extends nothing {
  int16 points = 0;
  void judger_reset() <= op judger.reset {
    points = 0;
} }
```

Assess review state over the system:

```
Assessment: ReviewOfComponentsStuff
query:      code review summary for chunk Components
```

reviewed	instancesJudging [InstanceConfiguration]
ready	Judge2 [AtomicComponent]
raw	testJudging [TestCase]
not reviewed yet	ContractMessages [MessageDefinitionTable]

Code Review and Security Audits

Track review state in the code

„Code“ includes requirements!

Use crypto to sign the review.

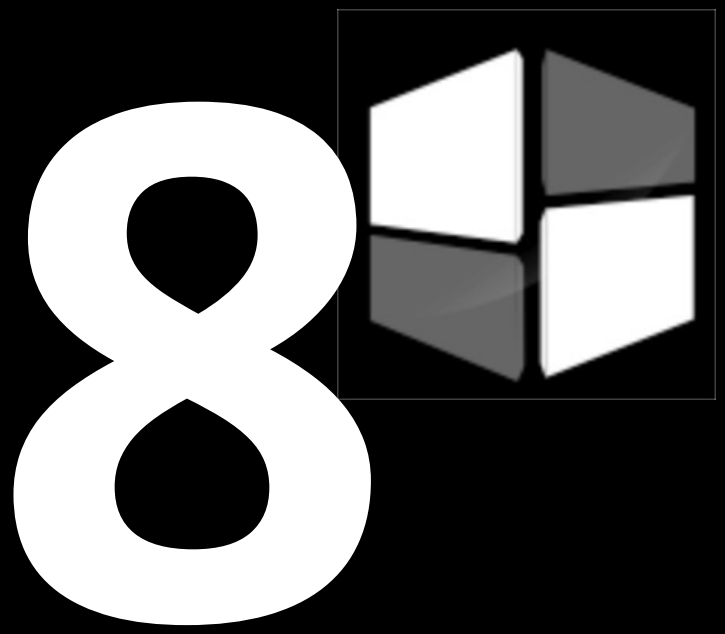
Branching-safe, because in the code.

Assess review state over the system:

Highlight unreviewed code.

On any granularity.

Can be exported into process-related tools.



Discussion

Evaluation Looks interesting, real evaluation is still missing.

Learning Languages have to be learned.

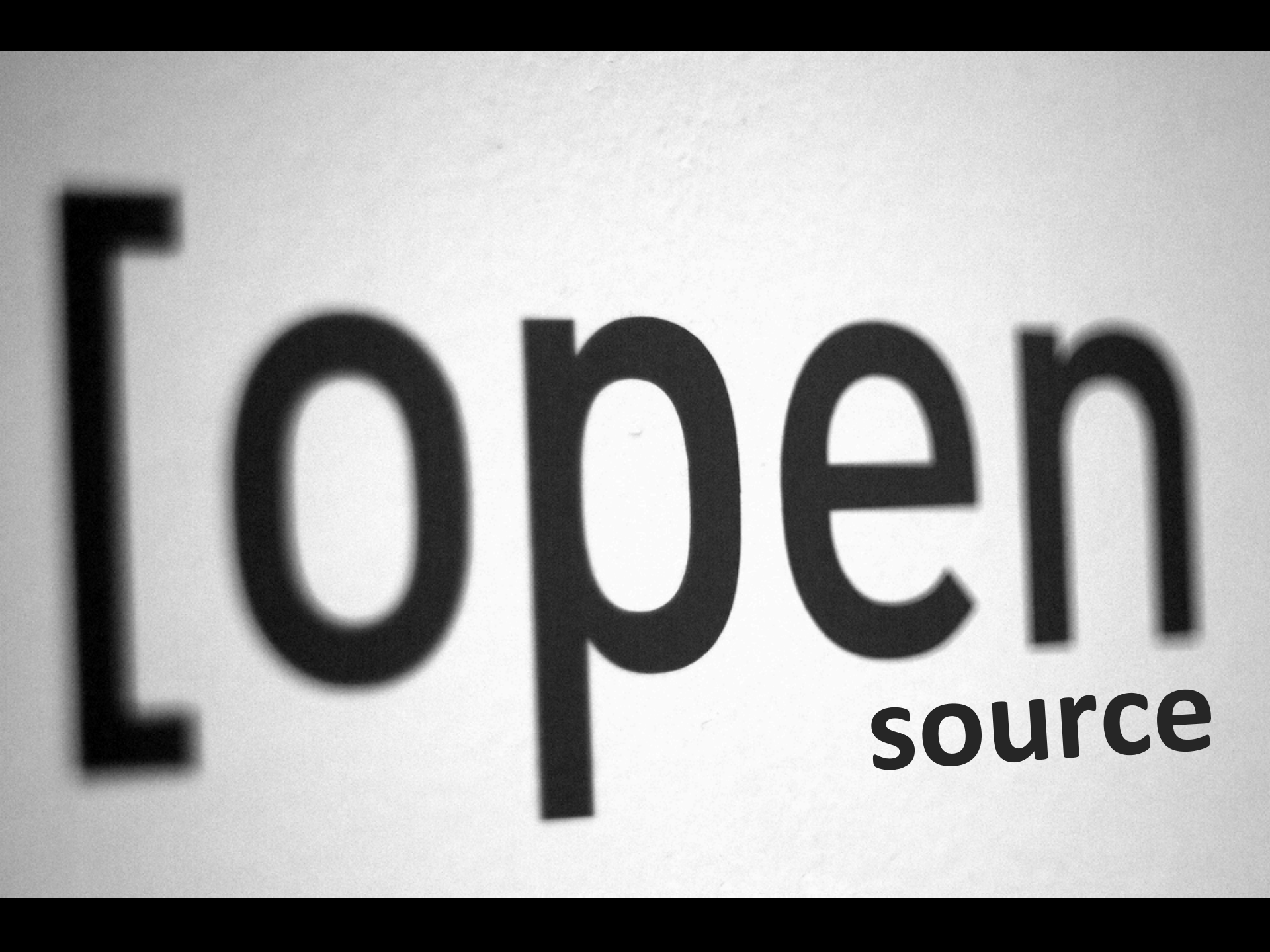
Developing MPS makes this very simple.

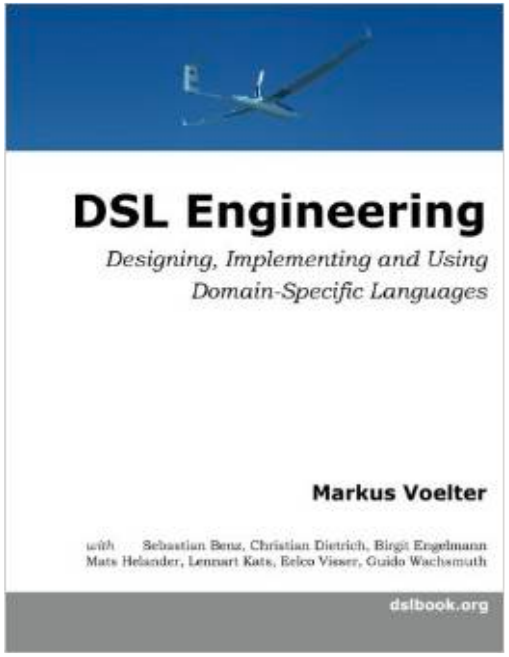
Trusting A tool qualification issue.

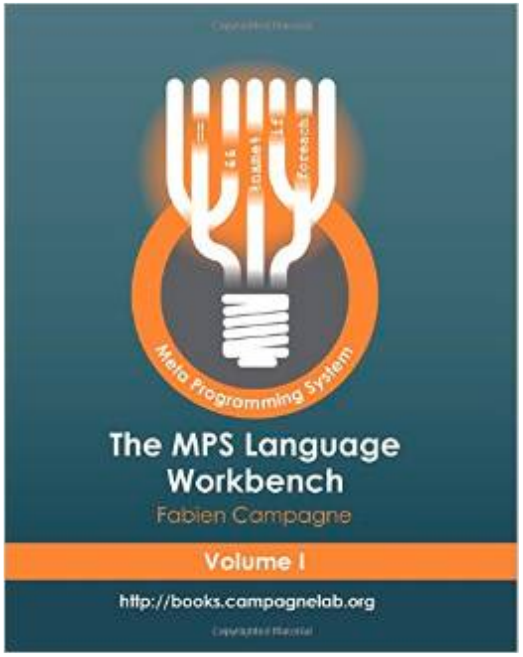
Legacy Code Import and Refactor.

Tool Lock-in Yes.

Other Languages Works in Principle.


open
source





mbeddr

MPS Meta Programming System

voelter { ingenieurbüro für softwaretechnologie // itemis

voelter@acm.org
www.voelter.de
@markusvoelter