Towards improving Software Security using Language Engineering and mbeddr

Markus Völter, Zaur Molotnikov, Bernd Kolb
1
Software Security
Software security refers to the security properties of a software system’s implementation.

G. McGraw,
Addison-Wesley, 2006.
Software security

- Techniques
  - Programming Language
    - (and the wrong use of it)

- Process
  - Education
  - Awareness
  - Reviews
Software security

Techniques

Programming Language

C
Software security

Techniques

Process

Programming Language

Make C better and less dangerous through suitable language extensions.
Software security

Techniques

Process

Make C better and less dangerous through suitable language extensions.
3

Language

Workbenches
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.
[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
MPS uses a Projectional Editor

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

```c
// [ 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

```latex
\text{double} \text{ midnight2(int32 a, int32 b, int32 c)} \{ \\
\quad \text{return } \frac{-b + \sqrt{b^2 - \sum_{i=1}^{4} a * c}}{2 * a} ;
\}
\text{ midnight2 (function)}
```

Tables

```plaintext
table
<table>
<thead>
<tr>
<th>alt</th>
<th>spd &gt; 0</th>
<th>spd &gt; 100</th>
<th>otherwise 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>= 0</td>
<td>10</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>&gt; 0</td>
<td>30</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>&gt; 100</td>
<td>50</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>
```

Graphical

```
```


[Projectional Editing] 
Advantage: Language Composition

Separate Files
Type System
Transformation
Constraints

In One File
Type System
Transformation
Constraints
Syntax
IDE

5+ base languages
50+ extensions to C
10+ extensions to requirements lang.
[Projectional Editing]
Advantage: Language Composition

No change to definition of $L_1$ or $L_2$ in order to use them together.

Embedding

$L_{Host} + L_{Adapt} + L_{Emb} = \text{Embedding}$

Extension

$L_{Base} + L_{Ext} = \text{Extension}$

Extension Composition

$L_{Base} + L_{Ext1} + L_{Ext2} = \text{Extension Composition}$
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

The State of the Art in Language Workbenches – LWC Conclusions
Systematic Comparison of different language workbenches and their features
An extensible set of integrated languages for embedded software engineering.
Open Source @ eclipse.org
Eclipse Public License 1.0
http://mbeddr.com
itemis  7 developers, project management
fortiss  2 developers, verification support
SIEMENS  1 developer, verification support
SIOUX    3 developers, C++

strategic collaboration with
Some of the C Extensions

Units

```c
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)
```

Components

```c
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
}
```

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

Math

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

```c
double midnight1(int32 a, int32 b, int32 c) {
  double d = b * b - 4 * a * c;
  if (d < 0) {
    return -1.0;
  } else {
    double r1, r2;
    r1 = (-b + sqrt(d)) / (2 * a);
    return r1;
  }
} 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 sollicitudin

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 a
It does so by taking into account
parameters: [int16 alt: current altitude of the trackpoint] =

result = (BASEPOINTS * 1) *

<table>
<thead>
<tr>
<th>speed</th>
<th>alt &gt; 2000</th>
<th>alt &gt; 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 180</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>&gt; 130</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

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 = beforeFlight
state beforeFlight {
  entry { points = 0; }
  on next [tp->alt > 0 m] -> airborne
    exit { points += TAKEOFF; } -> implements PointsForTakeoff
} state beforeFlight

Visualisations
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 physical units and you cannot or do not want to use generic units. An example is `anExistingFunction`:

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

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.

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

Code from `someFunction`
Solution: Technique
Code Markup and Checking

```c
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 $\subseteq$ Security

Layers checking, access permissions

„Units“ for sanitized or encrypted data
Straightforward Language Extensions

Apple GotoFail bug:

```c
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:

```c
trysequentially {
    validateStep1(data, ...);
    validateStep2(data, ...);
    validateStep3(data, ...);
} on fail (errorcode) {
    handleFailedValidation(data, errorcode, ...);
}
```
Better than macros:

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

Amazon’s s2n library

Danger:

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

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

```c
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.

```c
/* Returns 1 if a and b are equal, in constant time */
int s2n_constant_time_equalss(string, string, len);
```

Amazon’s s2n library

Naming conventions not checkable.
Semantic Annotations better.

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

Detect insecure C functions:

strncpy and the like.

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

Check this with tool.
Verification I

Heartbleed bug:

```c
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

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

---

```
// Parsing the message
size_t length = pM->payload_length;
uint8* p = pM->payload;

// Just some memory to read into, allocate
void* dest = malloc(length);

// Here a problem happens, because
memcpy(dest, p, length);
```
Verification III

Even better:

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

```c
// a field representing a timestamp
uint8[6] f_time = {0x00A, // first byte
                 UNIT_TIME24, // third byte
                 3, // 3 payload bytes
                 10, 20, 00} // last three bytes};

// 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 following
                    f_time, // embed the time field
                    f_value// embed the value field
                    ;
```

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

message ... { ... }
...
Solution: Process
Better Abstraction, Better Review

This is better than a switch-case:

```javascript
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
}
```

<table>
<thead>
<tr>
<th>States</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>timer(int64 t)</td>
</tr>
<tr>
<td></td>
<td>[t - tEnter &gt; 1000] -&gt; green { tEnter = t; }</td>
</tr>
<tr>
<td>green</td>
<td>buttonPressed()</td>
</tr>
<tr>
<td></td>
<td>[ ] -&gt; red { button = true; }</td>
</tr>
<tr>
<td></td>
<td>[t - tEnter &gt; 500] -&gt; red { tEnter = t; }</td>
</tr>
</tbody>
</table>
Better Notation, Better Review

```c
int32 averageIntArray(int32[] arr, int32 size) {
    return \sum_{i = 0}^{size} arr[i] / size;
}
```

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

```json
[core data] BillingRegion
- code [key]: string
- name: string
- baseMinPrice: float
- maxRebateFactor: float
```
Tracing

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

Analyze and find untraced program fragments.

Different kinds of traces to express semantics.

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

6.1 You should land as short as possible

6.2 Once you land successfully, you get another

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

<table>
<thead>
<tr>
<th>reviewed</th>
<th>instancesJudging [InstanceConfiguration]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ready</td>
<td>Judge2 [AtomicComponent]</td>
</tr>
<tr>
<td>raw</td>
<td>testJudging [TestCase]</td>
</tr>
<tr>
<td>not reviewed</td>
<td>ContractMessages [MessageDefinitionTable]</td>
</tr>
</tbody>
</table>
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]
Thank you!