# Adding C++ Support to mbeddr

## Language Engineering to Build an IDE for C++

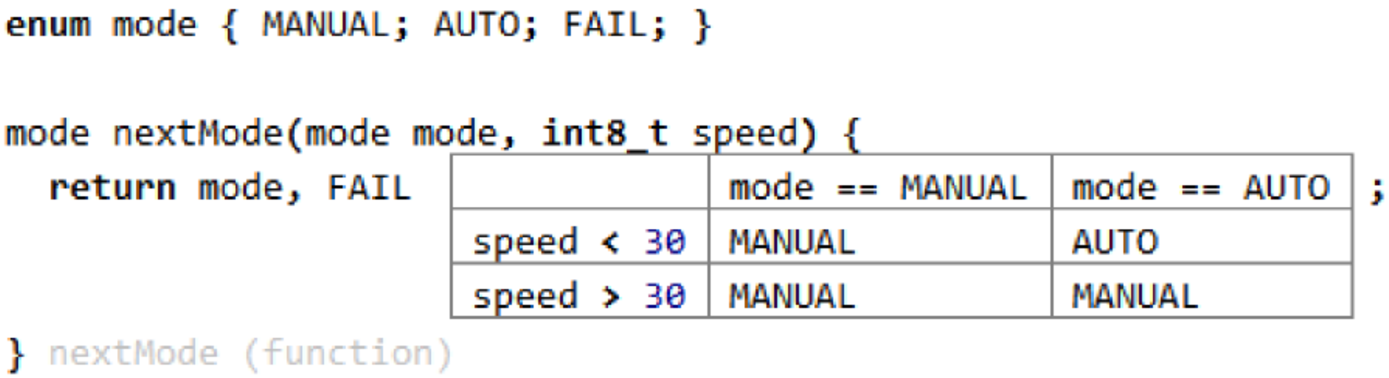
Master's Thesis

Presents: *Zaur Molotnikov*

Advisor: Dr. rer. nat. Daniel Ratiu
Supervisor: PD Dr. rer. nat. habil. Bernhard Schätz

# Context

# mbeddr : Decision Table



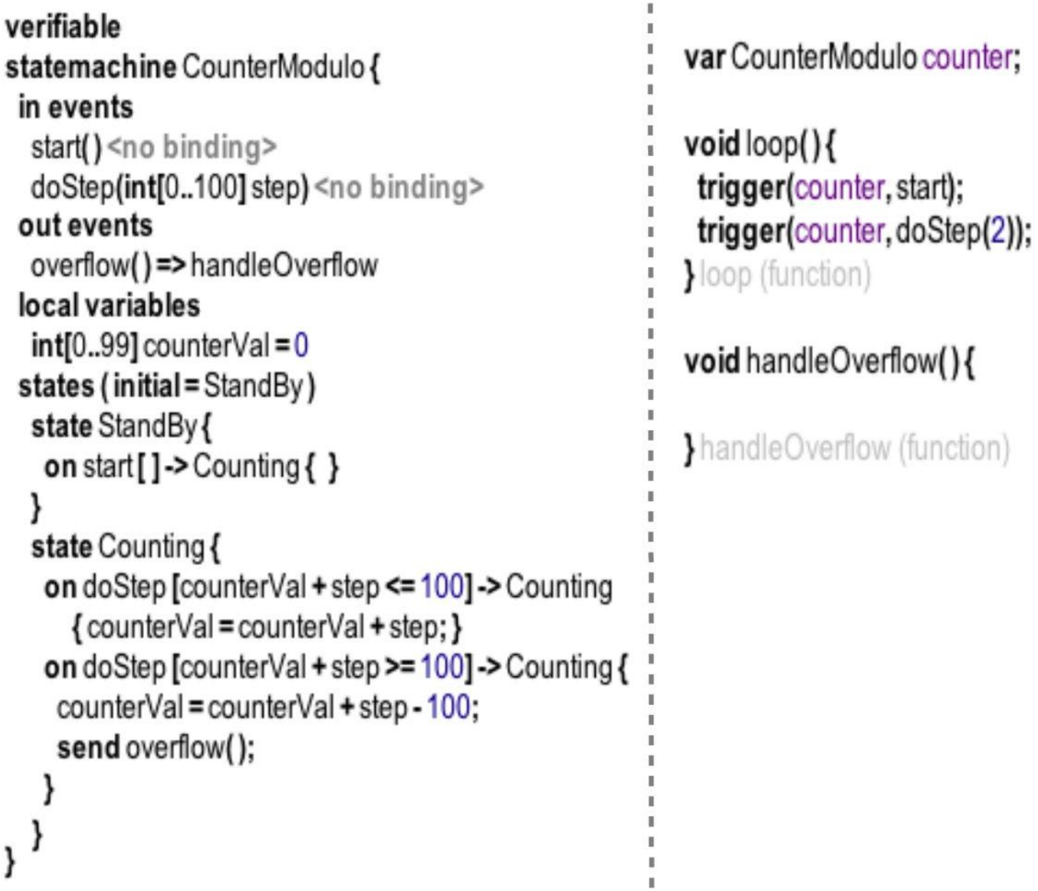
```
enum mode { MANUAL; AUTO; FAIL; }

mode nextMode(mode mode, int8_t speed) {
  return mode, FAIL |            | mode == MANUAL | mode == AUTO | ;
                    | speed < 30 | MANUAL         | AUTO         |
                    | speed > 30 | MANUAL         | MANUAL       |
} nextMode (function)
```

- A C function contains a decision table inside
- A higher-level construction than if..else cascade
- Features an analysis for completeness/conssitency

# mbeddr : State Machine

- C code contains a statemachine
- A higher-level construction than a set of variables and a long switch statement
- Features verification



```
verifiable
statemachine CounterModulo {
 in events
  start() <no binding>
  doStep(int[0..100] step) <no binding>
 out events
  overflow() => handleOverflow
 local variables
  int[0..99] counterVal = 0
 states ( initial = StandBy )
  state StandBy {
   on start [ ] -> Counting { }
  }
  state Counting {
   on doStep [counterVal + step <= 100] -> Counting
     { counterVal = counterVal + step; }
   on doStep [counterVal + step >= 100] -> Counting {
    counterVal = counterVal + step - 100;
    send overflow();
   }
  }
}
```

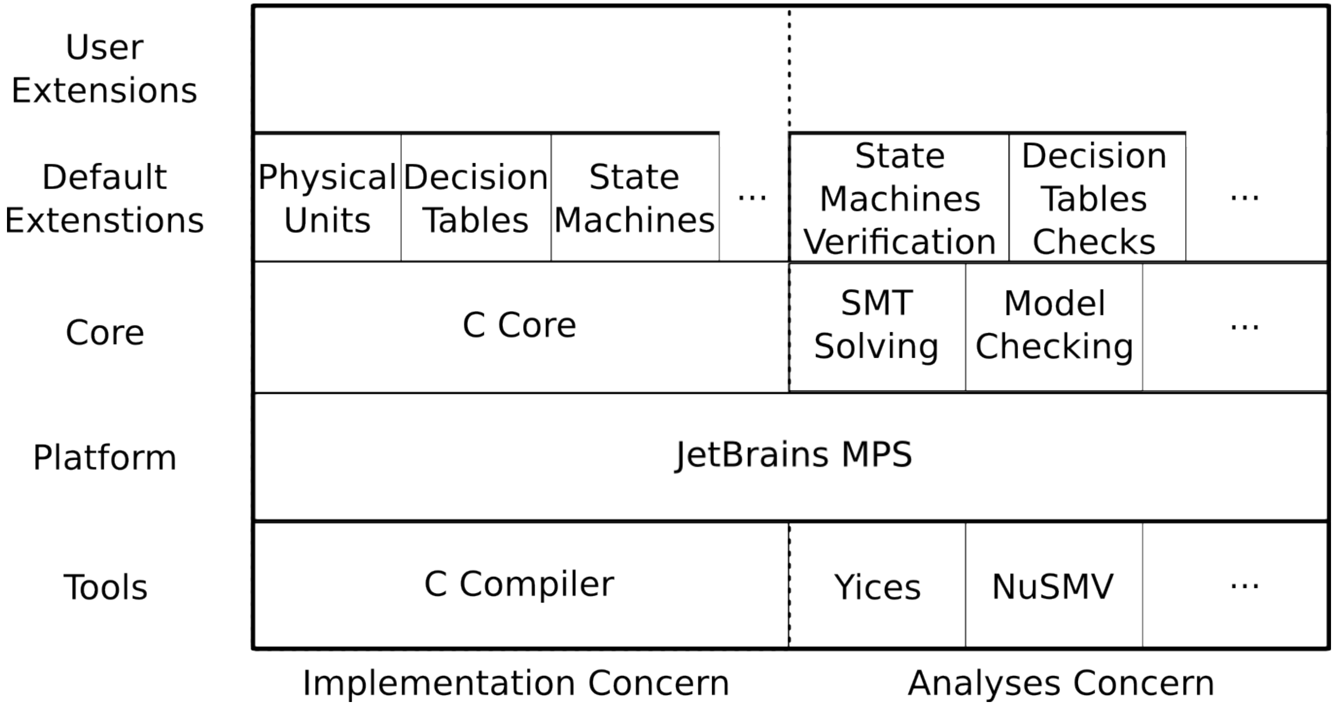
```
var CounterModulo counter;

void loop() {
 trigger(counter, start);
 trigger(counter, doStep(2));
} loop (function)

void handleOverflow() {

} handleOverflow (function)
```

# mbeddr : Technology Stack



| | Implementation Concern | Analyses Concern |
|---|---|---|
| User Extensions | | |
| Default Extenstions | Physical Units<br>Decision Tables<br>State Machines<br>... | State Machines Verification<br>Decision Tables Checks<br>... |
| Core | C Core | SMT Solving<br>Model Checking<br>... |
| Platform | JetBrains MPS | |
| Tools | C Compiler | Yices<br>NuSMV<br>... |

Safer C dialect + IDE for embedded development:

- only C core supported - “unsafe” constructions dropped
- domain specific extensions with analyses

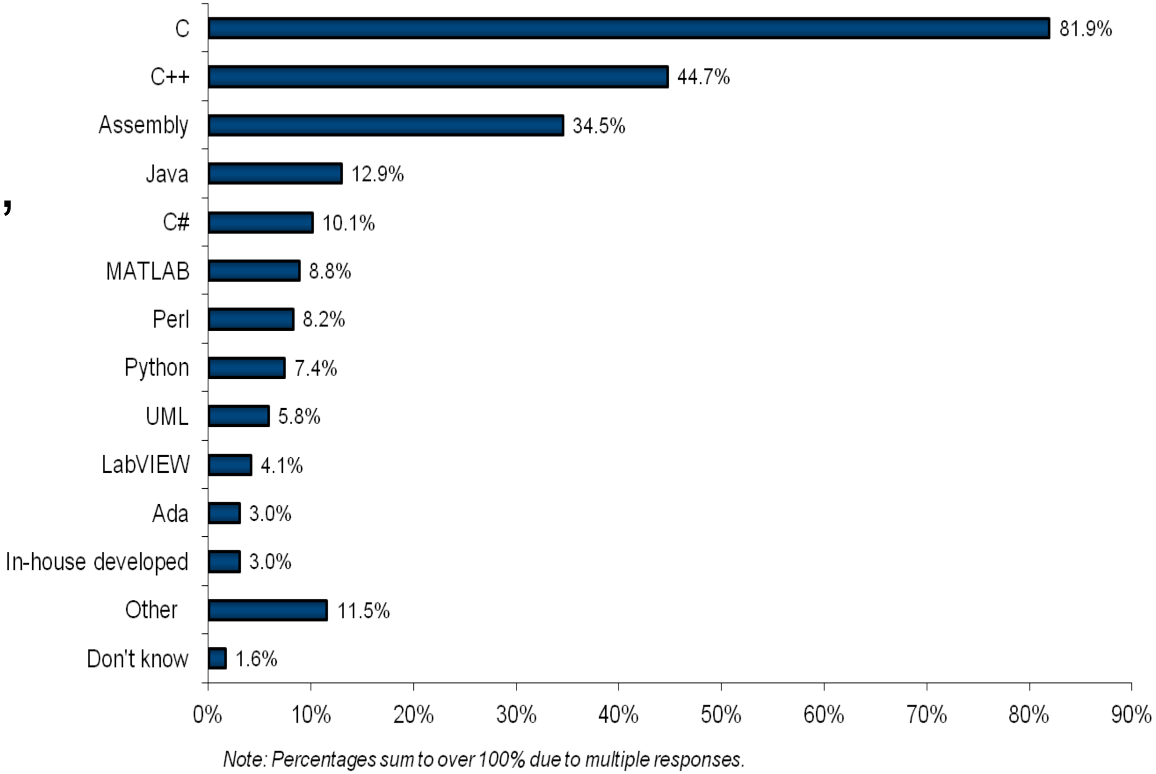
# Problem

# C++ Support

- C++ is popular among embedded system developers, but

- mbeddr does not support C++, so it makes sense to

- extend mbeddr to support C++



Note: Percentages sum to over 100% due to multiple responses.

Source - VDC Research: http://blog.vdcresearch.com/embedded_sw/2010/09/what-languages-do-you-use-to-develop-software.html

# Extending mbeddr

## Language Engineering in Practice

- **mbeddr core** is mainly a C programming language - all constructions are valid C++

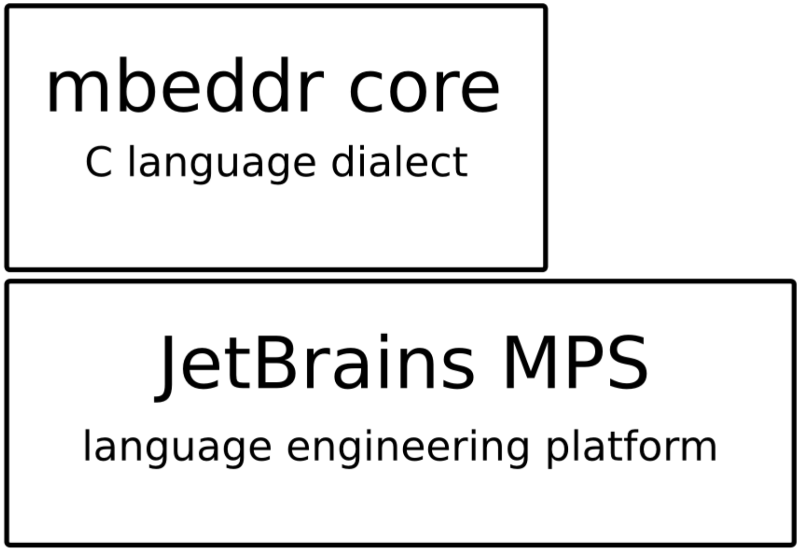
mbeddr core

C language dialect

# Extending mbeddr

## Language Engineering in Practice

- **mbeddr core** is mainly a C programming language - all constructions are valid C++
- based on a language engineering framework **JetBrains MPS**



mbeddr core

C language dialect

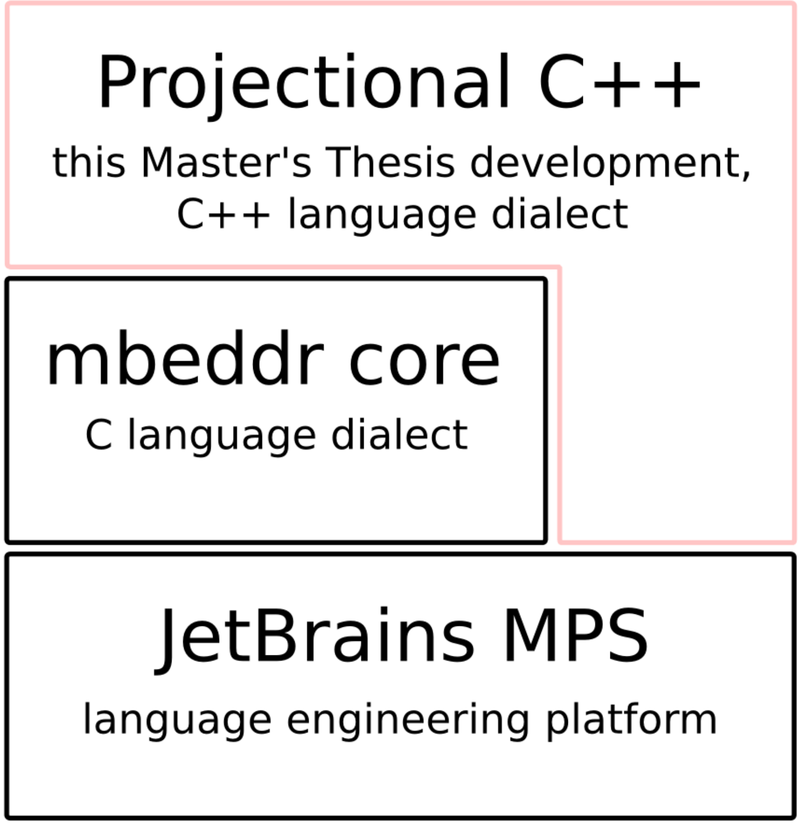
JetBrains MPS

language engineering platform

# Extending mbeddr

## Language Engineering in Practice

- **mbeddr core** is mainly a C programming language - all constructions are valid C++
- based on a language engineering framework **JetBrains MPS**
- to which we add C++ programming language, we call it ***Projectional C++***



Projectional C++

this Master's Thesis development, C++ language dialect

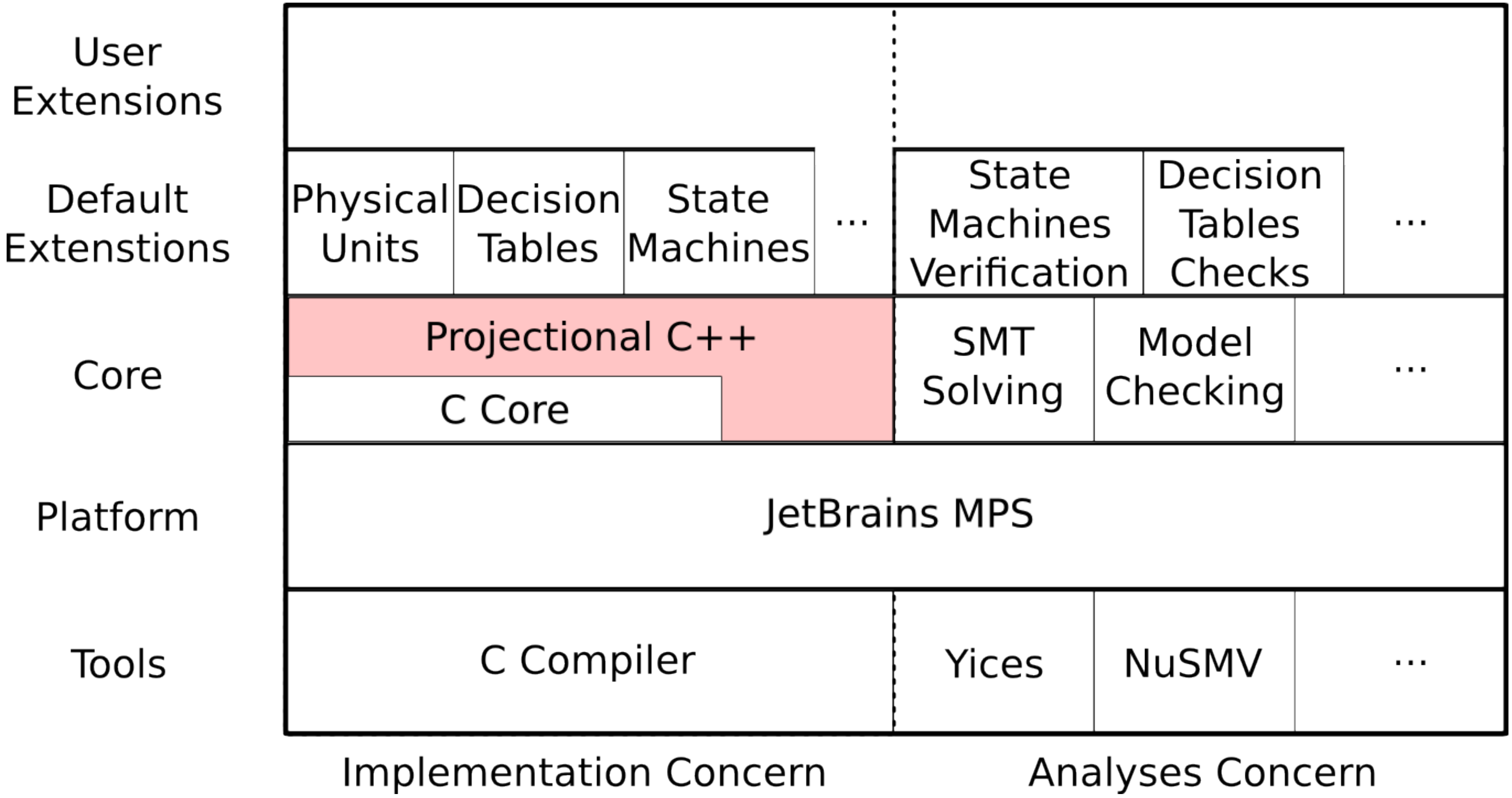
mbeddr core

C language dialect

JetBrains MPS

language engineering platform

# Projectional C++ in mbeddr Technology Stac



User Extensions

Default Extenstions

Physical Units

Decision Tables

State Machines

...

State Machines Verification

Decision Tables Checks

...

Core

Projectional C++

C Core

SMT Solving

Model Checking

...

Platform

JetBrains MPS

Tools

C Compiler

Yices

NuSMV

...

Implementation Concern

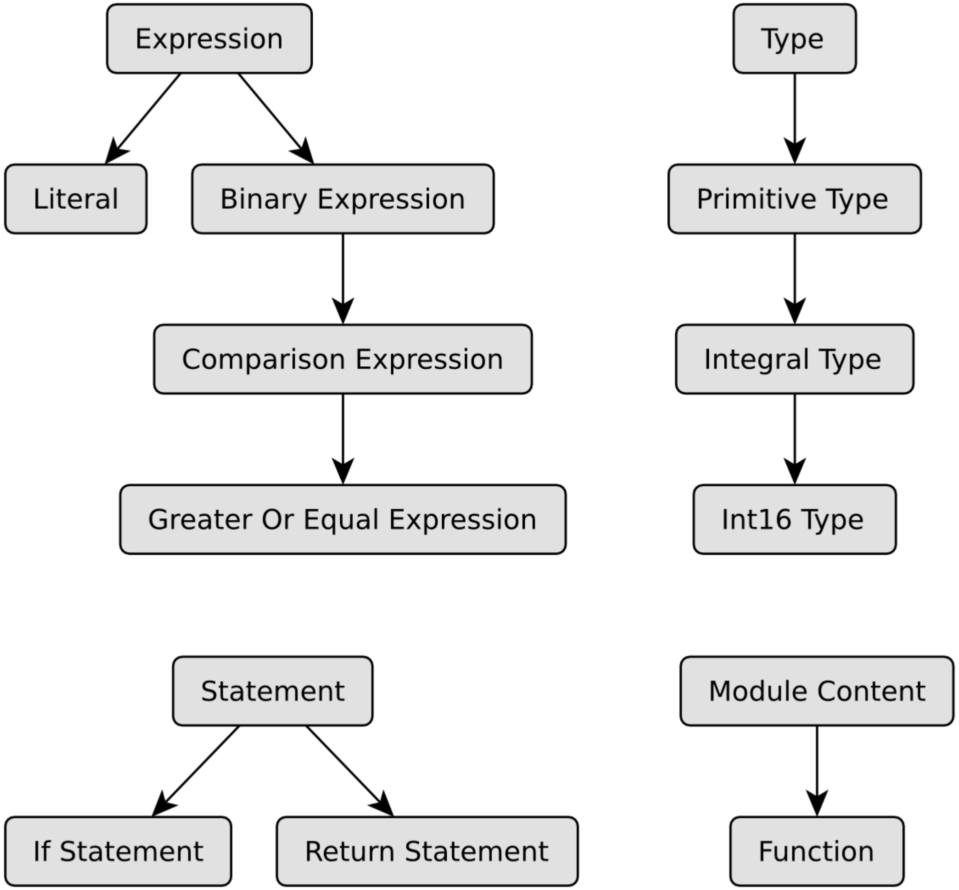
Analyses Concern

# Approach

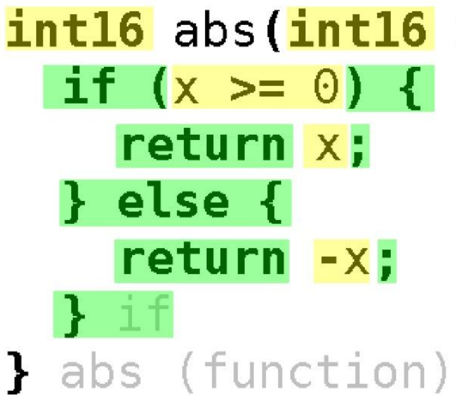
# Meta-Model Hierarchies

```
int16 abs(int16 x) {
  if (x >= 0) {
    return x;
  } else {
    return -x;
  } if
} abs (function)
```

- Language syntax is a meta-model
- Model is the code
- Code is *projected*



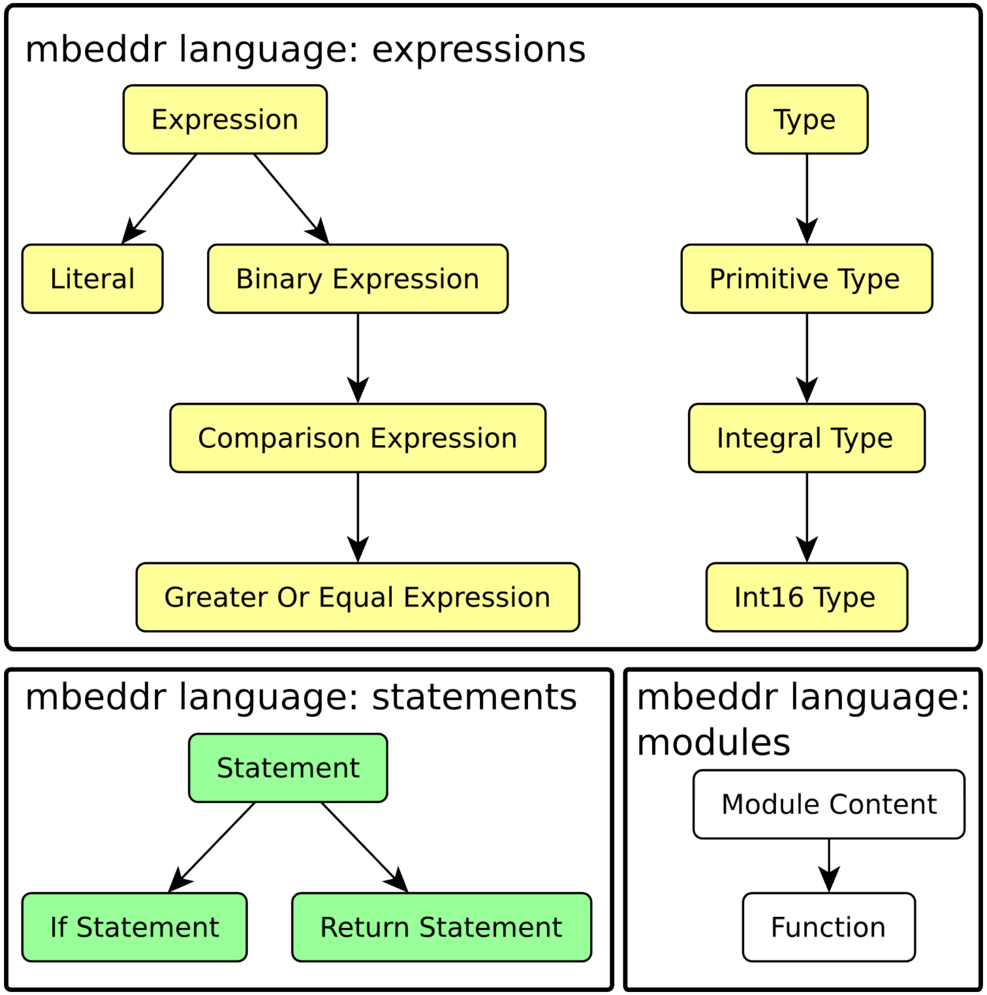
# Language Modularity



```
int16 abs(int16 x) {
  if (x >= 0) {
    return x;
  } else {
    return -x;
  } if
} abs (function)
```

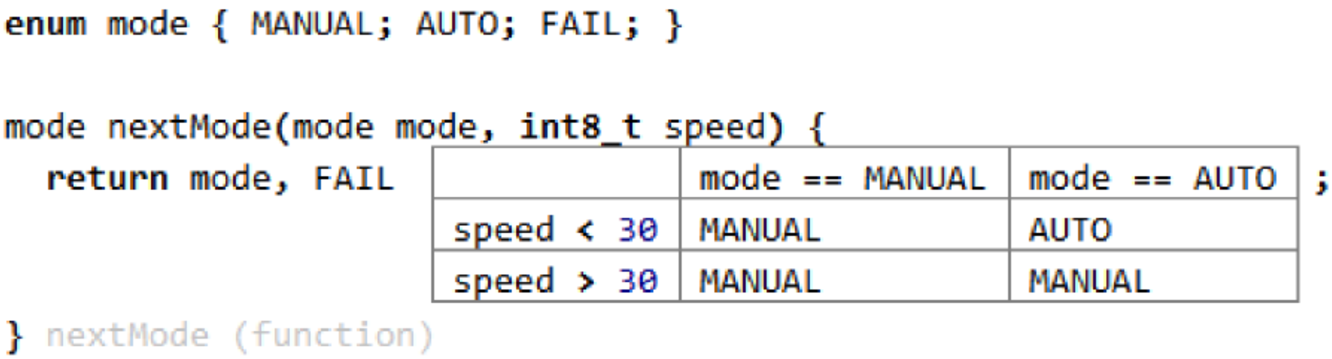
- statements language uses expressions

- modules language uses expressions and statements languages



# Language Extensibility

- state machines language extends expressions language



```
enum mode { MANUAL; AUTO; FAIL; }

mode nextMode(mode mode, int8_t speed) {
  return mode, FAIL
```

| | mode == MANUAL | mode == AUTO |
|---|---|---|
| speed < 30 | MANUAL | AUTO |
| speed > 30 | MANUAL | MANUAL |

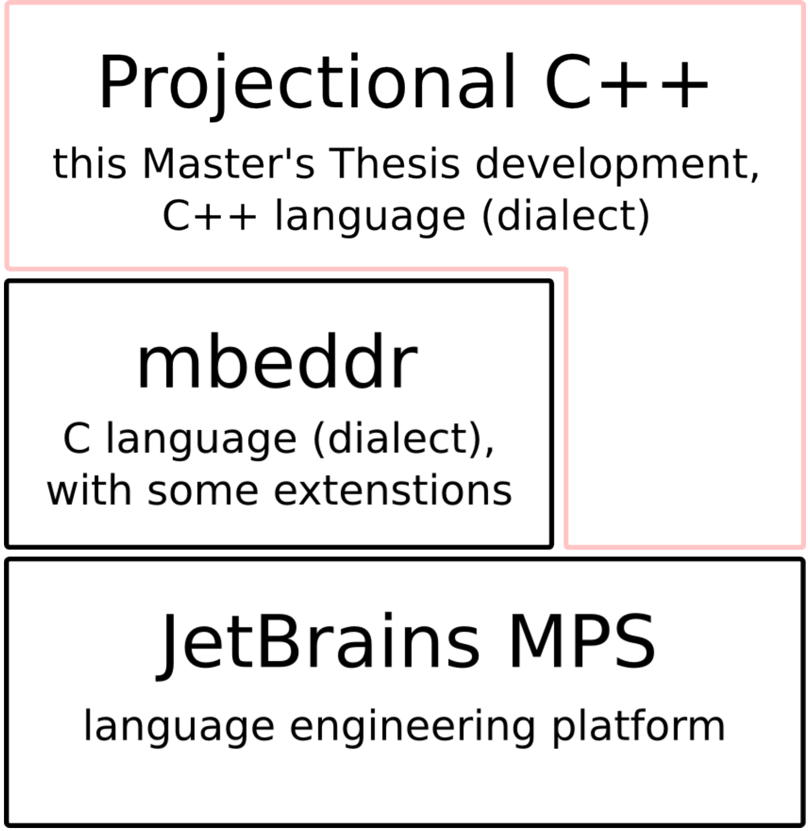
```
;
} nextMode (function)
```

# Views on a Language

- A language is defined in views on it:
  - **Structure** view - meta-model structure
  - **Behavior** view - methods for nodes, like in a class
  - **Editor** view - the way to input and edit a model
  - **Constraints** view - context-sensitive limitations
  - **Type system** view - for typed languages
  - **Analyses** view - for warnings and errors, informing
  - **Generators** view - used for cascade generation
  - **TextGen** view - to generate a model to text
  - **Intentions** view - provide user-callable automations

# Approach

- Add C++ constructions to mbeddr C language
- describing a new language in JetBrains MPS through *views* on it,
- with the use of *language modularity* and
- *language extensibility*.



Projectional C++

this Master's Thesis development, C++ language (dialect)

mbeddr

C language (dialect), with some extenstions

JetBrains MPS

language engineering platform

# Contribution

# Practical Challenges

**C1:** Is it in general possible to extend mbeddr C to C++? *Will mbeddr be flexible enough?*

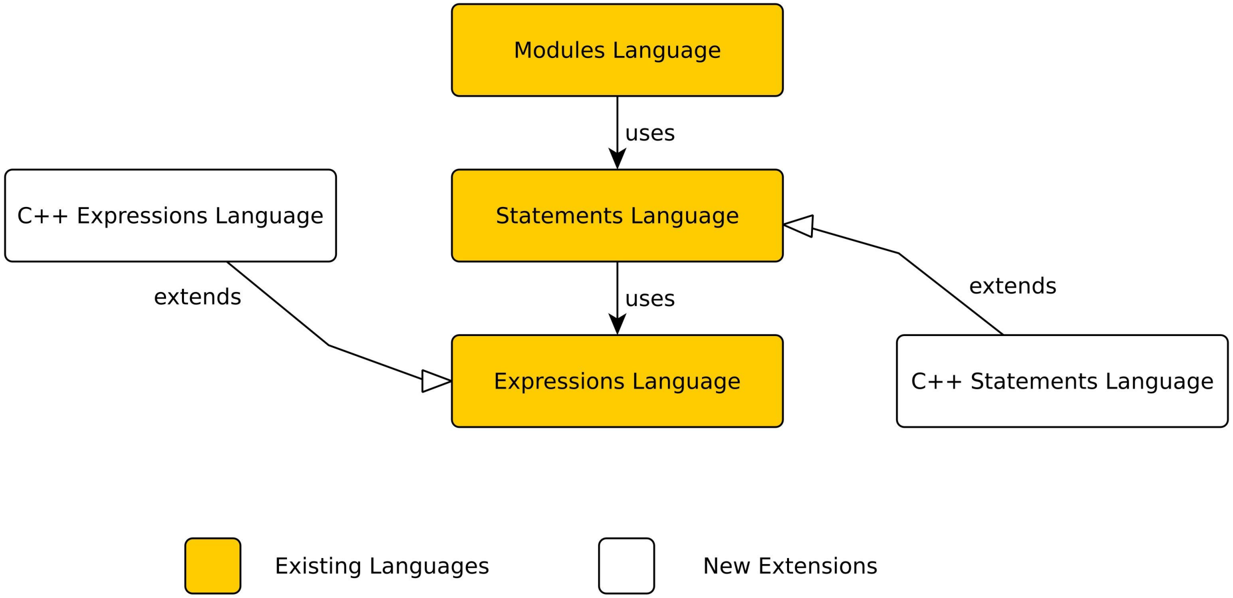
**C2:** Is it possible to make a “better” C++ dialect? *Like mbeddr C is a “better” C dialect.*

**C3:** Templates in C++ bear pure textual nature! *A contradiction with the projectional approach.*

*C1 - C3  are Challenges 1 - 3.

# C1: Extending C to C++

- Practically proven to be possible
  - One-side-awareness challenge: mbeddr should not be aware of Projectional C++

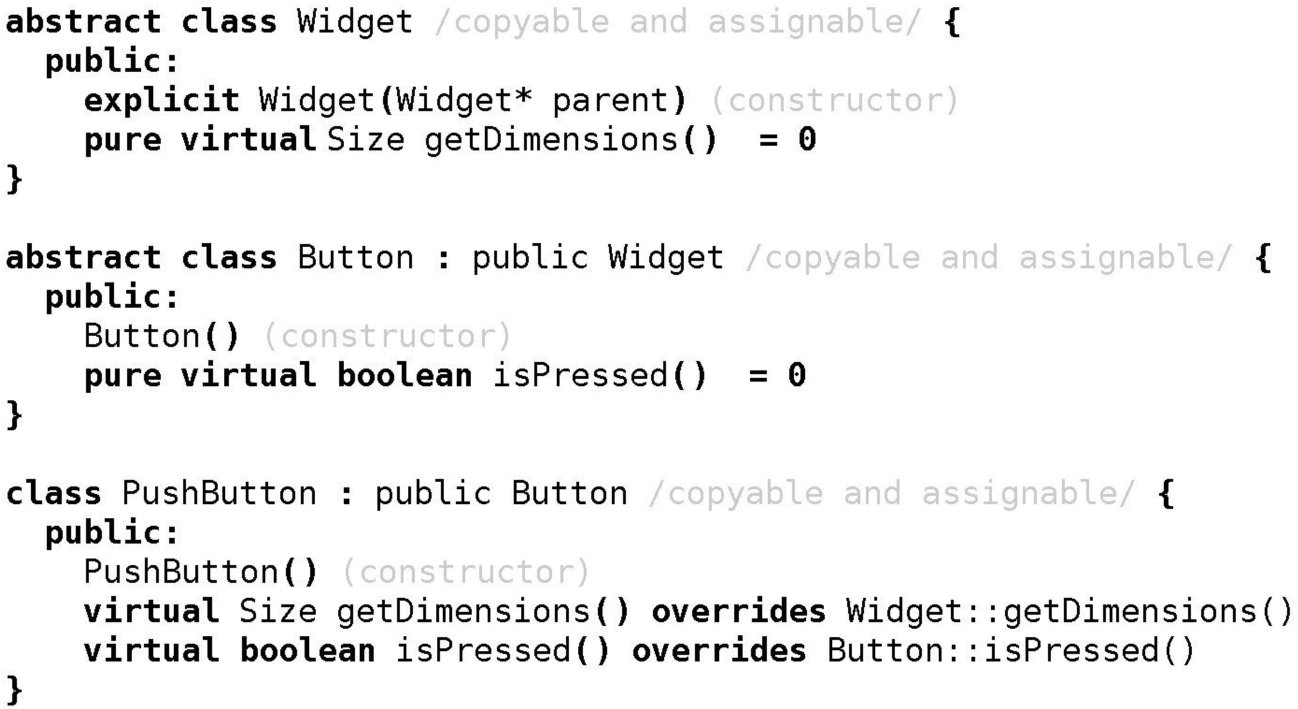


# C2: “Better” C++ Dialect?

- **Projectional C++ is extensible**
  - Potential extensions: signals, design patterns, more?
- **“No” to dropping language features**
  - C++ is valuable with the standard library (STL), but
  - STL requires *all* C++ language features, thus
  - dropping “unsafe” language features is not the way!
- **Added language features**
  - Analyses to improve understanding *(abstract class)*
  - Information, made explicit *(override)*
  - Code generation, automations *(getter and setter)*
  - Naming conventions made explicit *(naming of fields*

# Adding Features to C++

- Abstract classes, pure virtual functions and overrides have no explicit syntax in C++, added:



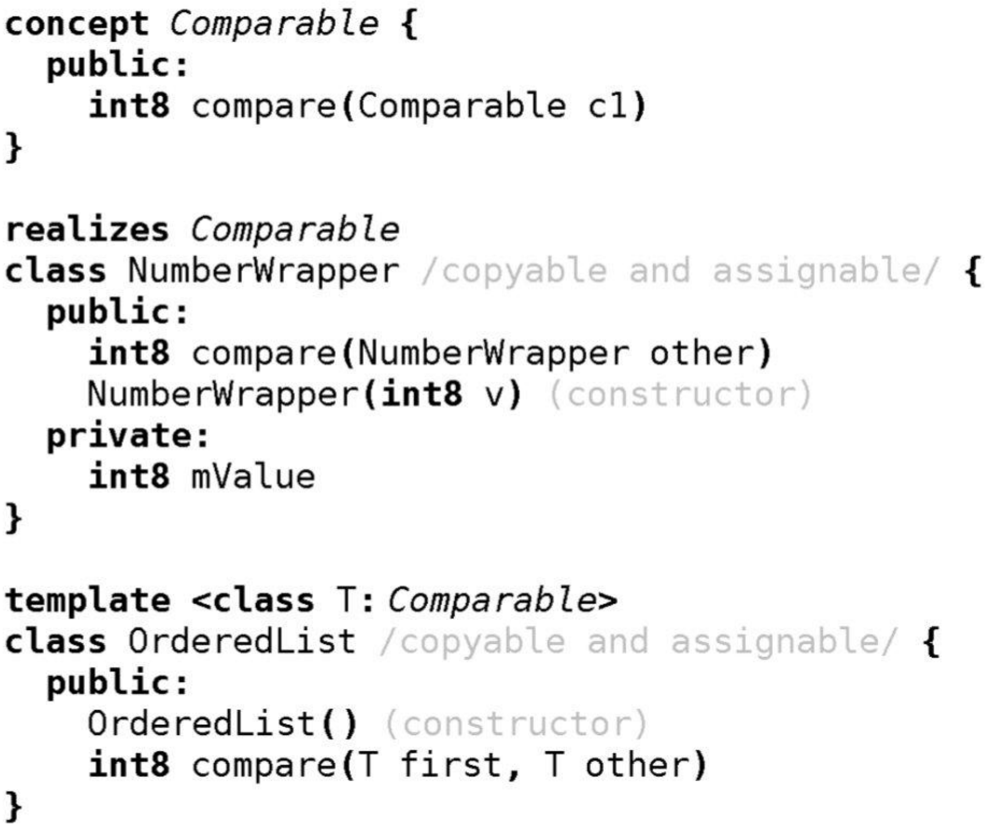
```
abstract class Widget /copyable and assignable/ {
  public:
    explicit Widget(Widget* parent) (constructor)
    pure virtual Size getDimensions()  = 0
}

abstract class Button : public Widget /copyable and assignable/ {
  public:
    Button() (constructor)
    pure virtual boolean isPressed()  = 0
}

class PushButton : public Button /copyable and assignable/ {
  public:
    PushButton() (constructor)
    virtual Size getDimensions() overrides Widget::getDimensions()
    virtual boolean isPressed() overrides Button::isPressed()
}
```

# C3: Templates?

- Implemented through “C++ concepts”



```
concept Comparable {
  public:
    int8 compare(Comparable c1)
}

realizes Comparable
class NumberWrapper /copyable and assignable/ {
  public:
    int8 compare(NumberWrapper other)
    NumberWrapper(int8 v) (constructor)
  private:
    int8 mValue
}

template <class T: Comparable>
class OrderedList /copyable and assignable/ {
  public:
    OrderedList() (constructor)
    int8 compare(T first, T other)
}
```

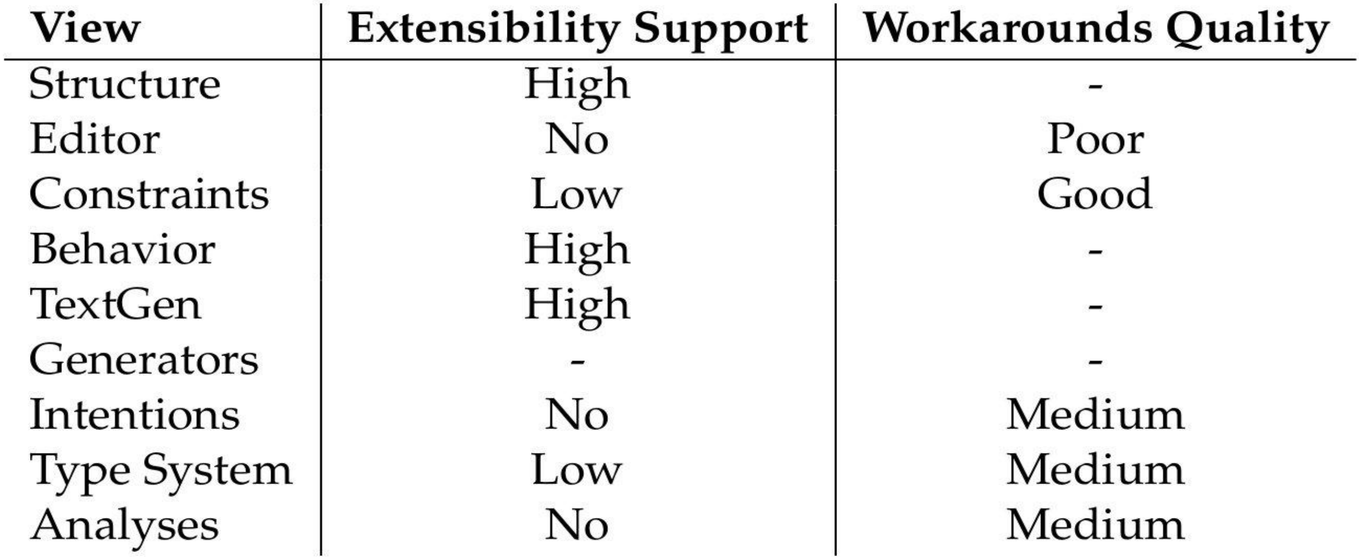
# Templates - Discussion

- Advantages of C++ concepts approach:
  - requirements on template parameter are explicit
  - and checkable

- Disadvantages of C++ concepts approach:
  - the feature is absent in C++ as it is
  - special importer needed to extract concepts
  - additional user work when creating template code
  - potential code duplication of a new nature

# Lessons Learned

# Meta-Model Extensibility



| View | Extensibility Support | Workarounds Quality |
|---|---|---|
| Structure | High | - |
| Editor | No | Poor |
| Constraints | Low | Good |
| Behavior | High | - |
| TextGen | High | - |
| Generators | - | - |
| Intentions | No | Medium |
| Type System | Low | Medium |
| Analyses | No | Medium |

- MPS design defines language extensibility
- MPS could provide a better support for it

# Making a Language Safer

Few principles discovered may apply to every language reconstructed:

- Target semantics - no focus on syntax for a parser
- Store more information - like overrides
- Configuration is a part of source - like naming
- Hide redundant syntax - like braces, etc.
- Make syntax human readable - like pure virtuals
- Show core, hint on details - like friend function
- Perform analyses - preventive and informative

# Language Tooling

- Analyses were found to be useful, however
  - MPS does not support them explicitly!
  - Computational complexity can be very high!

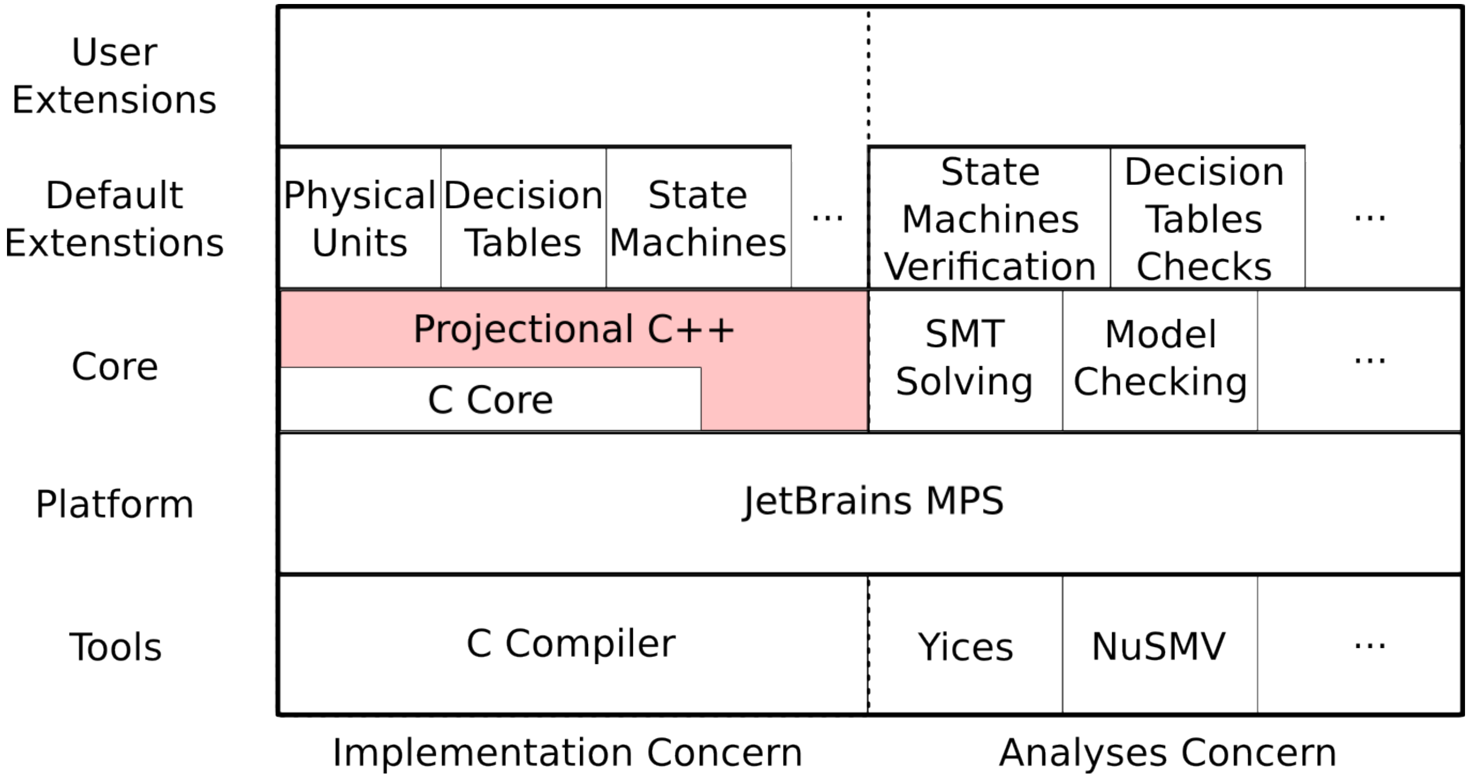
- Propositions for MPS evolution - APIs for analyses:
  - When does an analysis start?
  - Which scope does it have?
  - Is result caching needed?
  - Prioritisation, concurrency limitations?
  - Informing the user - can be improved and
  - Common solutions offered for reuse

# Future Work

- Complete language support - C++ is large
- STL implementation - in projection, import?
- Investigating language use - in practice
- Importer, templates - for existing text code
- Debugger - for C++ constructions
- Extensions - signals, patterns, more?
- MPS Evolution - ways proposed to JetBrains

# Thank you for attention!

## Questions are welcome!



User Extensions

Default Extensions: Physical Units | Decision Tables | State Machines | ... || State Machines Verification | Decision Tables Checks | ...

Core: Projectional C++ | C Core || SMT Solving | Model Checking | ...

Platform: JetBrains MPS

Tools: C Compiler || Yices | NuSMV | ...

Implementation Concern | Analyses Concern

**Zaur Molotnikov**, zaur@zaurmolotnikov.com