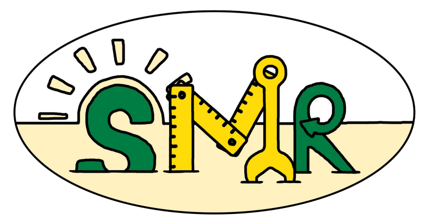
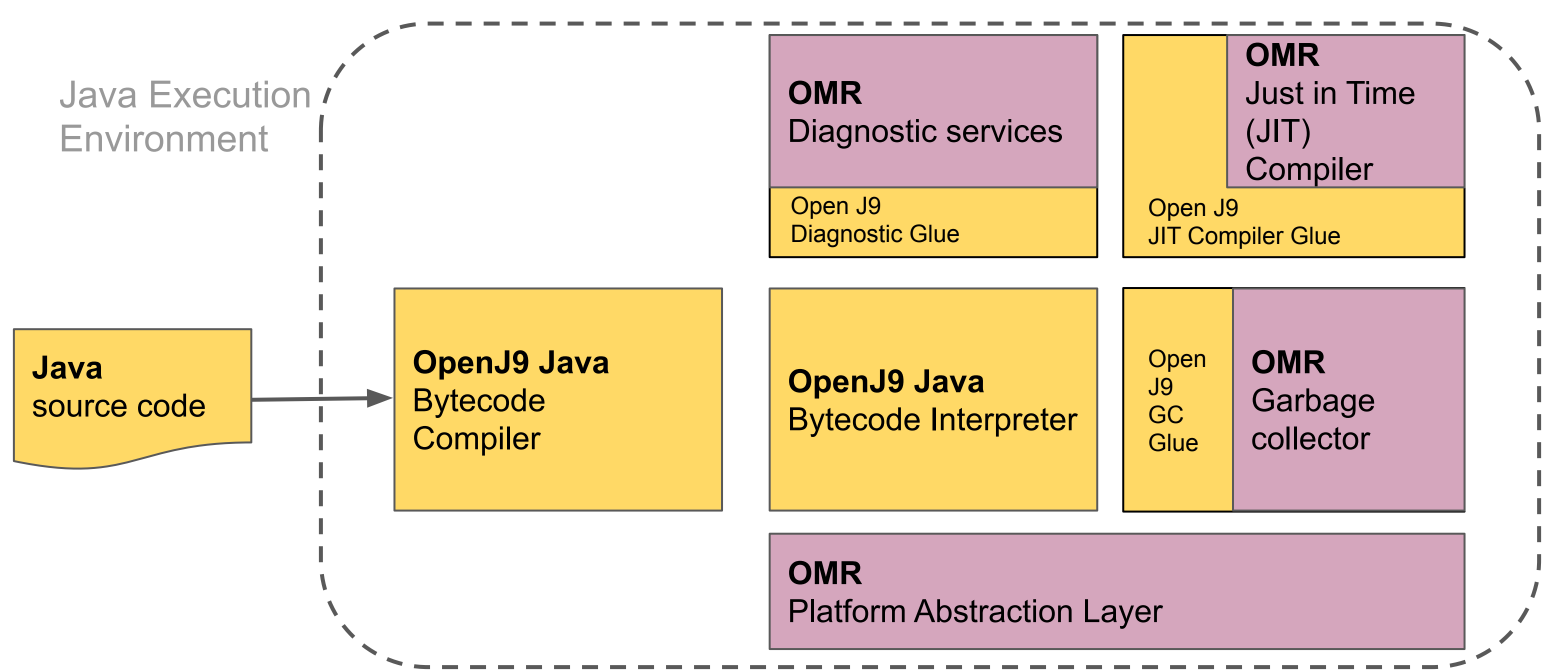
Tackling Variability Implementation Challenges in Eclipse OMR

Batyr Nuryyev, Sarah Nadi, Leonardo Banderali
{nuryyev, nadi}@ualberta.ca
leob@ibm.com



Eclipse OMR

Eclipse OMR is a set of **reusable** C++ components for building language runtimes such as **JIT compiler** and **garbage collector.**



Variability in Eclipse OMR

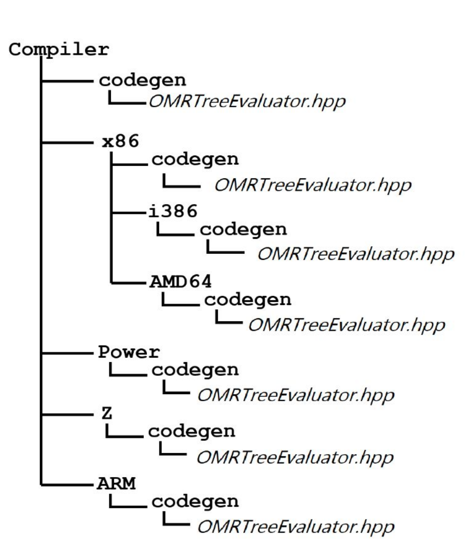
Languages
(Java, Python, etc.)

Architectures
(x86, ARM, Power, Z)

OMR’s current variability mechanism in Compiler component

OMR's variability mechanism

Static polymorphism/extensible classes



OMR::TreeEvaluator

typedef

Extends Extends Extends Extends

OMR::X86::TreeEvaluator | OMR::Z::TreeEvaluator | OMR::P::TreeEvaluator | OMR::ARM::TreeEvaluator

Extends Extends typedef typedef typedef typedef

OMR::X86::i386::TreeEvaluator | OMR::X86::AMD64::TreeEvaluator

typedef typedef

<<typedef>>
OMR::TreeEvaluatorConnector

Extends

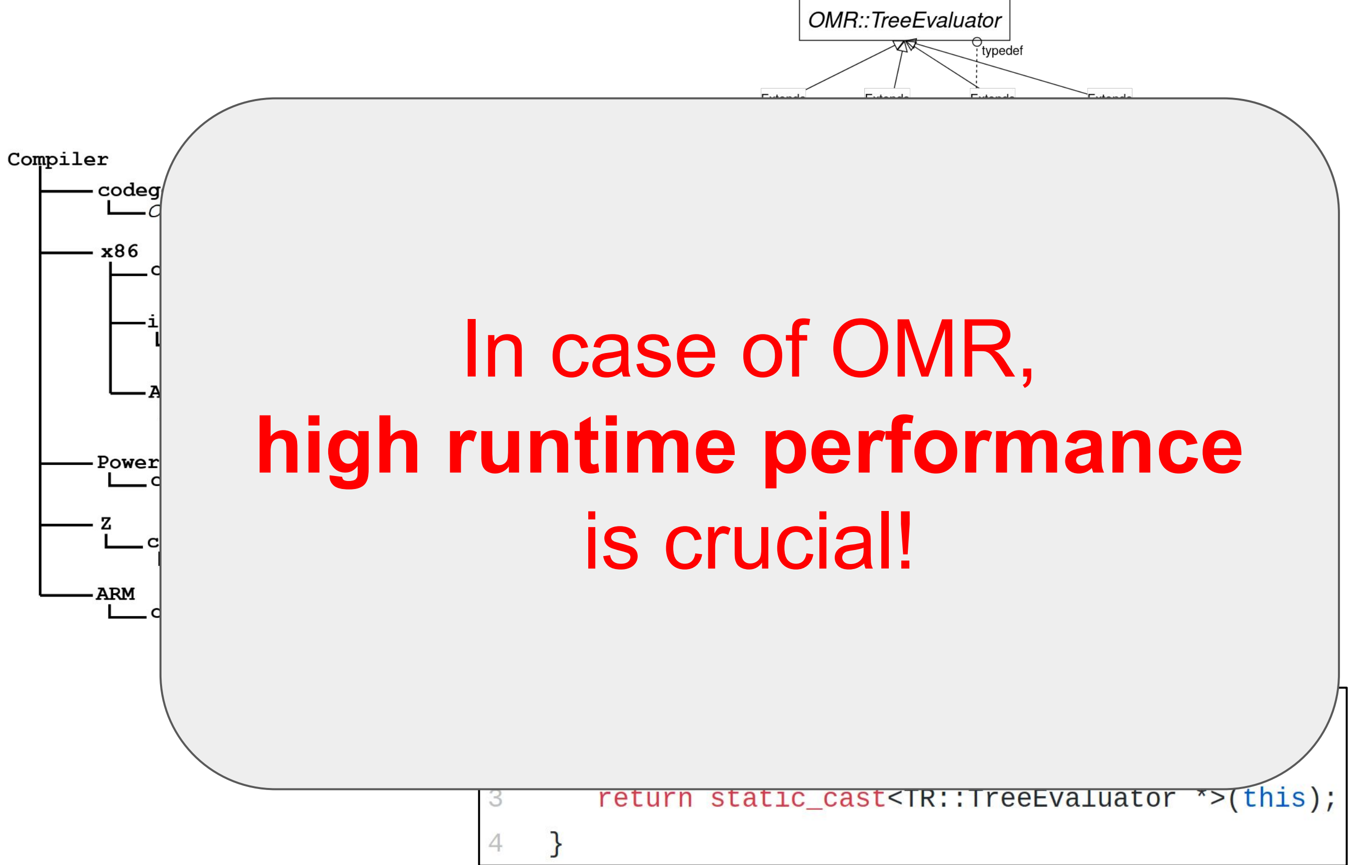
lang::TreeEvaluator

Extends

TR::TreeEvaluator

```
1  TR::TreeEvaluator * OMR::TreeEvaluator::self()
2  {
3     return static_cast<TR::TreeEvaluator *>(this);
4  }
```

OMR’s variability mechanism
Static polymorphism/extensible classes



Project Goal

Help OMR developers understand variability in their code and investigate variability implementation alternatives

2017-2018: Static polymorphism is a root of all problems

Static polymorphism

2017-2018: Static polymorphism is a root of all problems

Too complex

Static polymorphism

2017-2018: Static polymorphism is a root of all problems

Too complex

Static polymorphism

We tried to switch to **dynamic polymorphism**

Dynamic polymorphism

SPLC 2018

Using Static Analysis to Support Variability Implementation Decisions in C++

Samer AL Masri, Sarah Nadi
University of Alberta
AB, Canada
{almasrinadi}@ualberta.ca

Matthew Gaudet*
Mozilla
Ottawa, ON, Canada
mgaudet@mozilla.com

Xiaoli Liang, Robert W. Young
IBM Canada
Markham, ON, Canada
{xsliang,rwyoung}@ca.ibm.com

CASCON 2017

Software Variability Through C++ Static Polymorphism

A Case Study of Challenges and Open Problems in Eclipse OMR

Samer AL Masri[†], Nazim Uddin Bhuiyan[†], Sarah Nadi[†], Matthew Gaudet[‡]
University of Alberta[†], IBM Canada[‡]
{almasri,nazimudd,nadi}@ualberta.ca,magaudet@ca.ibm.com

Dynamic polymorphism

SPLC 2018

Using Static Analysis to Support Variability Implementation Decisions in C++

Samer AL Masri, Sarah Nadi
University of Alberta
AB, Canada
{almasrinadi}@ualberta.ca

Matthew Gaudet*
Mozilla
Ottawa, ON, Canada
mgaudet@mozilla.com

Xiaoli Liang, Robert W. Young
IBM Canada
Markham, ON, Canada
{xsliang,rwyoung}@ca.ibm.com

CASCON 2017

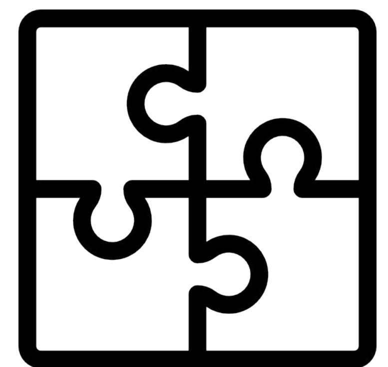
Software Variability Through C++ Static Polymorphism

A Case Study of Challenges and Open Problems in Eclipse OMR

Samer AL Masri[†], Nazim Uddin Bhuiyan[†], Sarah Nadi[†], Matthew Gaudet[‡]
University of Alberta[†], IBM Canada[‡]
{almasri,nazimudd,nadi}@ualberta.ca,magaudet@ca.ibm.com

However, there are constraints and requirements that dynamic polymorphism cannot resolve.

In **2018-2019,** We took a step back to get a **complete picture of all the current challenges**.



Icon made by Freepik from www.flaticon.com

Our Goal (2018-2020)

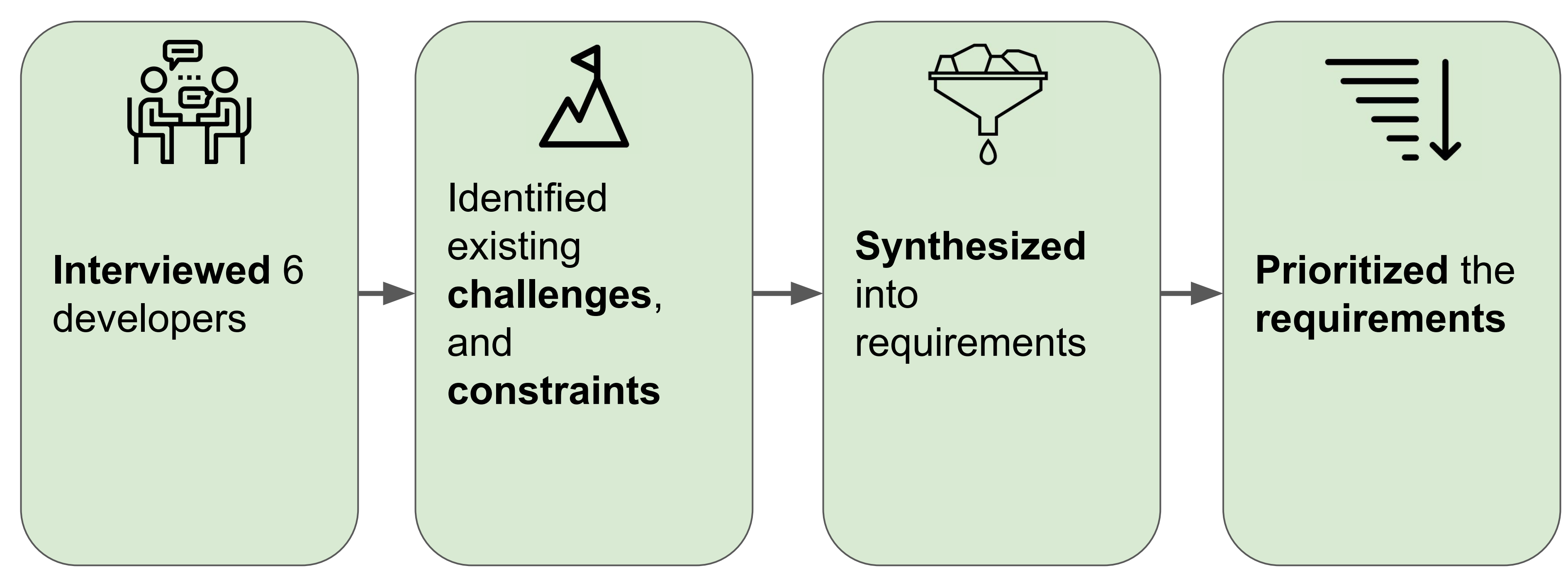
Identify current design challenges and explore any design alternatives.

Steps

I. Identifying requirements
    A. Example: C++ enum extensibility

II. Exploring solutions
    A. Literature exploration
    B. Industrial case studies

Step I: **Identifying Requirements**



Interview icon made by Becris from www.flaticon.com
Challenge icon made by Larea from www.thenounproject.com
Synthesis icon made by Deivid Sáenz from www.thenounproject.com
Priorities icon made by Phạm Thanh Lộc from www.thenounproject.com

Challenges, constraints, and requirements

Simplicity and **usability**

Mechanism for **extending C++ enums/unions**

No strongly connected components (e.g., templates)

Enable **finer control** over extension points

More **streamlined consistency checks**

Varying constructors across archs/langs

All requirements are listed here: https://youtu.be/F7FIE1QIUAE

Challenges, constraints, and requirements

Simplicity and **usability**

Mechanism for **extending C++ enums/unions**

No strongly connected components (e.g., templates)

Enable **finer control** over extension points

More **streamlined consistency checks**

Varying constructors across archs/langs

All requirements are listed here: https://youtu.be/F7FIE1QIUAE

Example:

C++ Enum/Union Extensibility Problem

Requirements Example
C++ Enum/Union Extensibility Problem

Opcodes.hpp

```
// enum values
iconst, // load int const
lconst, // load long const
fconst, // load float const
```

```
enum ILOpCodes
{
 #include "il/Opcodes.hpp"
 extraOpcode1
 extraOpcode2
};
```

OpcodeEnum.hpp

OpcodeProps.hpp

```
   // Opcode A
   {
   /* .name         = */ "iconst",
   /* .properties1 = */
ILProp1::LoadConst,
   /* other props ...*/
   },

   // Opcode B
   {
   /* .name       =   */ "lconst",
      ...
   },
// ...
```

Requirements Example

C++ Enum/Union Extensibility Problem

Opcodes.hpp

```
// enum values
iconst, // load int const
lconst, // load long const
fconst, // load float const
```

```
enum ILOpCodes
{
 #include "il/Opcodes.hpp"
 extraOpcode1
 extraOpcode2
};
```

OpcodeEnum.hpp

OpcodeProps.hpp

```
   // Opcode A
   {
   /* .name          = */ "iconst",
   /* .properties1 = */
ILProp1::LoadConst,
   /* other props ...*/
   },

   // Opcode B
   {
   /* .name        =    */ "lconst",
      ...
   },
// ...
```

Requirements Example
C++ Enum/Union Extensibility Problem

Opcodes.hpp

```
    // enum values
    iconst, // load int const
    lconst, // load long const
    fconst, // load float const
```

OpcodeEnum.hpp

```
enum ILOpCodes
{
 #include "il/Opcodes.hpp"
 extraOpcode1
 extraOpcode2
};
```

OpcodeProps.hpp

```
    // Opcode A
    {
    /* .name        = */ "iconst",
    /* .properties1 = */
ILProp1::LoadConst,
    /* other props ...*/
    },

    // Opcode B
    {
    /* .name      =    */ "lconst",
       ...
    },
// ...
```

Requirements Example
C++ Enum/Union Extensibility Problem

Opcodes.hpp

```
// enum values
iconst, // load int const
lconst, // load long const
fconst, // load float const
```

```
enum ILOpCodes
{
 #include "il/Opcodes.hpp"
 extraOpcode1
 extraOpcode2
};
```

OpcodeEnum.hpp

==

OpcodeEnum.hpp after preprocessing

```
enum ILOpCodes
{
 iconst, // load int const
 lconst, // load long const
 fconst, // load float const
 extraOpcode1
 extraOpcode2
};
```

Requirements Example

C++ Enum/Union Extensibility Problem

OMRILOpCodesEnum.hpp

```
//  NOTE: IF you add opcodes or change the order then you must fix the
following
//      files (at least): ./ILOpCodeProperties.hpp
//                         compiler/ras/Tree.cpp (2 tables)
//                         compiler/optimizer/SimplifierTable.hpp
//                         compiler/optimizer/ValuePropagationTable.hpp
//                         compiler/x/amd64/codegen/TreeEvaluatorTable.cpp
//                         compiler/x/i386/codegen/TreeEvaluatorTable.cpp
//                         compiler/p/codegen/TreeEvaluatorTable.cpp
//                         compiler/z/codegen/TreeEvaluatorTable.cpp
//                         compiler/aarch64/codegen/TreeEvaluatorTable.cpp
//                         compiler/arm/codegen/TreeEvaluatorTable.cpp
//                         compiler/il/OMRILOpCodesEnum.hpp
//                         compiler/il/ILOpCodes.hpp
// Also check tables in ../codegen/ILOps.hpp
```

Requirements Example

C++ Enum/Union Extensibility Problem

...MRILOpCodesEnum.hpp

```
//  NOTE: IF you ad... opcodes or c...                hen y... ust fix the
following
//       files (at le...
//
//                                                          .hpp
//                                                        nTable.hpp
//                                                         orTable.cpp
//                                                       uatorTable.cpp
//                                                        Table.cpp
//                                                          le.cpp
//                                                TreeEvaluat... Table.cpp
//                            ...egen/          EvaluatorTable.cpp
//                             ...LOpCodesEnum.hpp
//                     ...omp.../il/IL...pCodes.hpp
// Also check table...  ../codegen/ILOps.hpp
```

No C++ way of extending unions/enums, which leads to extending opcodes **confusing** and **more error-prone**.

Step II: Exploring solutions

Study existing mechanisms from the software variability literature

Exploring alternatives

Parameters

Design patterns

Feature-oriented Programming

Fine-grained → *Coarse-grained*

+ *Compile-time*, *load-time*, and *run-time*

Existing mechanisms: **Parameters**

```
1 class Conf {
2   public static boolean COLORED = true;
3   public static boolean WEIGHTED = false;
4 }
```

```
41    void print() {
42      if (Conf.COLORED)
43        Color.setDisplayColor(color);
44      System.out.print(id);
45    }
```

Reference: Sven Apel et al. 2013. *Feature-Oriented Software Product Lines: Concepts and Implementation.* Springer Publishing Company, Incorporated.

Existing mechanisms: **Parameters**

Pros

- Easy to use and understand
- Most languages *naturally* support this mechanism (*if* statements)

Cons

- Creates code bloat
- Adds run-time overhead

Existing mechanisms: **Parameters**

Pros

- Easy to use and understand
- Most languages *naturally* support this mechanism (*if* statements)

Cons

- Creates code bloat
- Adds run-time overhead

In OMR

✅ Simple and easy to understand

❌ Does not address all requirements (e.g., enum extensions, need for varying constructors)

Existing mechanisms: **Design patterns**

Pros

- Well known patterns
- Disciplined guidelines

Cons

- Pre-planning is necessary
- Boilerplate code

Existing mechanisms: **Design patterns**

Pros

- Well known patterns
- Disciplined guidelines

Cons

- Pre-planning is necessary
- Boilerplate code

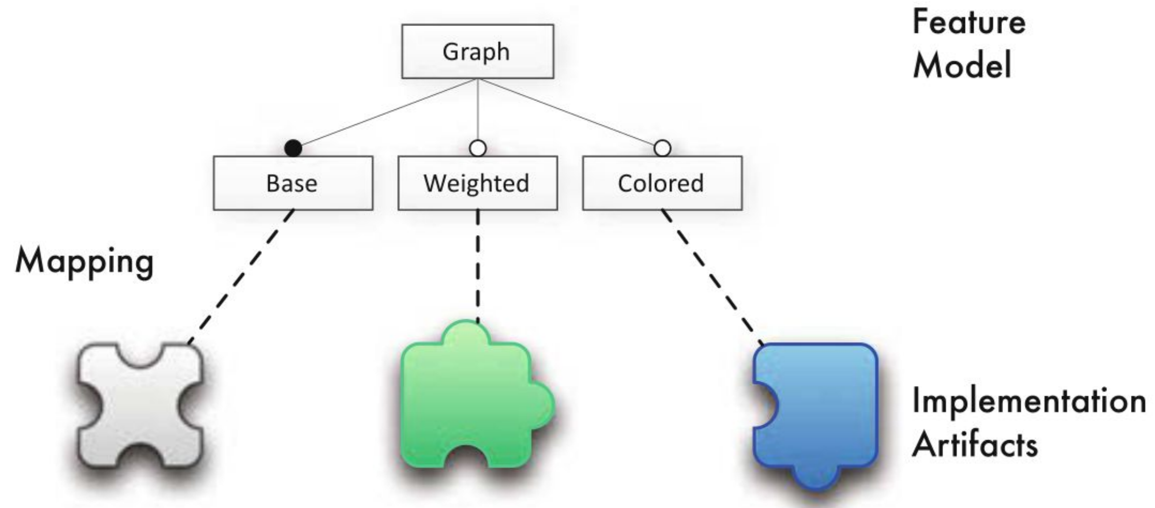
In OMR

✅ Facilitates communication between developers

❌ Requires massive refactoring effort

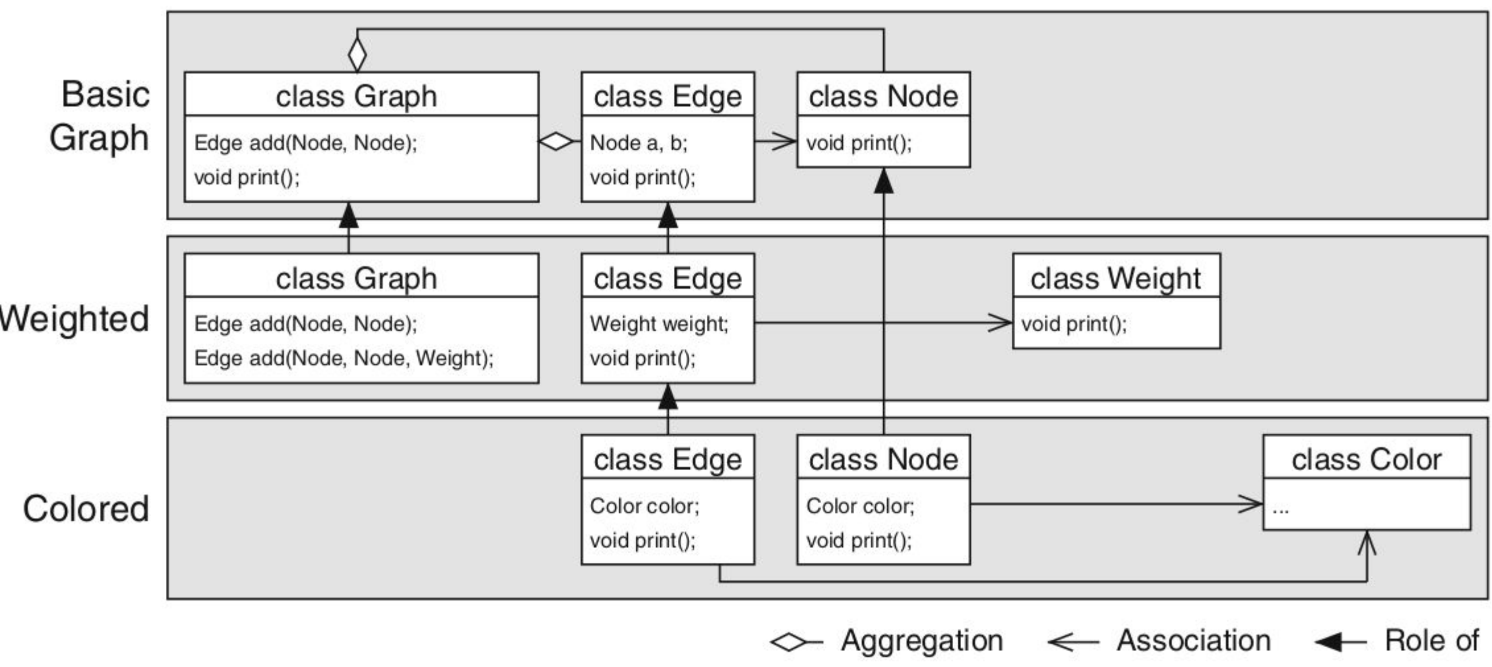
❌ Does not address all requirements (e.g., enum extensions)

Existing mechanisms: **Feature-oriented Programming**



Reference: Sven Apel et al. 2013. *Feature-Oriented Software Product Lines: Concepts and Implementation.* Springer Publishing Company, Incorporated.

Existing mechanisms: **Feature-oriented Programming**



Reference: Sven Apel et al. 2013. *Feature-Oriented Software Product Lines: Concepts and Implementation.* Springer Publishing Company, Incorporated.

Existing mechanisms: **Feature-oriented Programming**

```
1  layer BasicGraph;
2
3  class Graph {
4    Vector nodes = new Vector();
5    Vector edges = new Vector();
6    Edge add(Node n, Node m) {
7      Edge e = new Edge(n, m);
8      nodes.add(n);
9      nodes.add(m);
10     edges.add(e);
11     return e;
12   }
13   void print() {
14     for(int i = 0; i < edges.size(); i++) {
15       ((Edge)edges.get(i)).print();
16       if(i < edges.size() - 1)
17         System.out.print(" , ");
18     }
19   }
20 }
```

Reference: Sven Apel et al. 2013. *Feature-Oriented Software Product Lines: Concepts and Implementation.* Springer Publishing Company, Incorporated.

Existing mechanisms: **Feature-oriented Programming**

```
1  layer BasicGraph;
2
3  class Graph {
4    Vector nodes = new Vector();
5    Vector edges = new Vector();
6    Edge add(Node n, Node m) {
7      Edge e = new Edge(n, m);
8      nodes.add(n);
9      nodes.add(m);
10     edges.add(e);
11     return e;
12   }
13   void print() {
14     for(int i = 0; i < edges.size(); i++) {
15       ((Edge)edges.get(i)).print();
16       if(i < edges.size() - 1)
17         System.out.print(" , ");
18     }
19   }
20 }
```

```
1  layer Weighted;
2
3  refines class Graph {
4    Edge add(Node n, Node m) {
5      Edge e = Super.add(n, m);
6      e.weight = new Weight();
7      return e;
8    }
9    Edge add(Node n, Node m, Weight w) {
10     Edge e = add(n, m);
11     e.weight = w;
12     return e;
13   }
14 }
```

Reference: Sven Apel et al. 2013. *Feature-Oriented Software Product Lines: Concepts and Implementation.* Springer Publishing Company, Incorporated.

Existing mechanisms: **Feature-oriented Programming**

Pros

- Good feature traceability
- Separation of concerns

Cons

- Only academic tools so far
- Requires tool support

Existing mechanisms: **Feature-oriented Programming**

Pros

- Good feature traceability
- Separation of concerns

Cons

- Only academic tools so far
- Requires tool support

In OMR

✅ Makes it simpler to track features

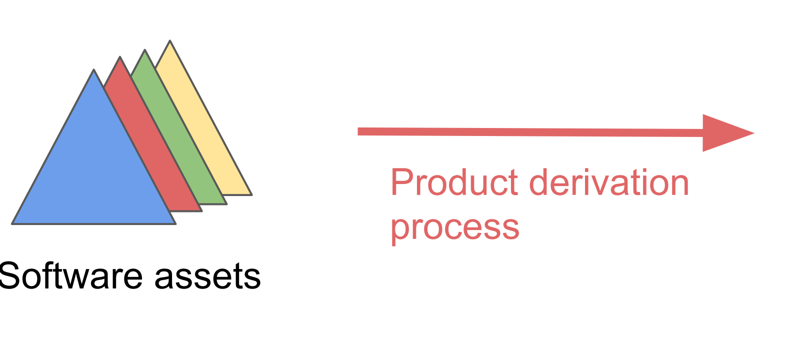
❌ Requires a complete rehaul

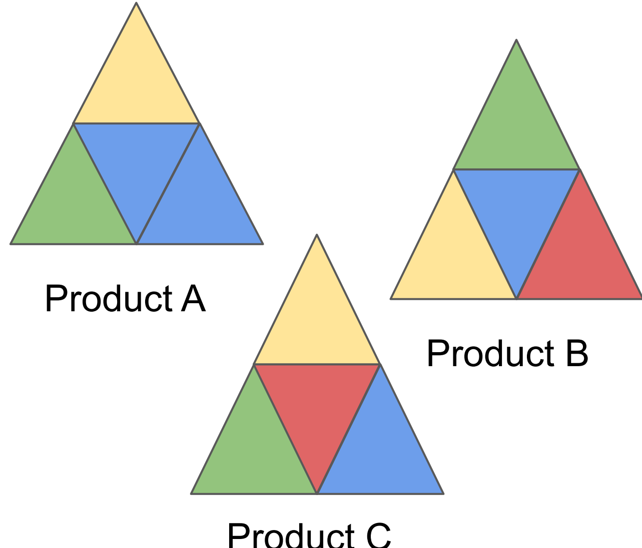
❌ Does not meet all requirements (e.g., enum extensions)

Industrial case studies

Industrial case studies

Most industry case studies focus on **extracting a software product line** from a set of end products.



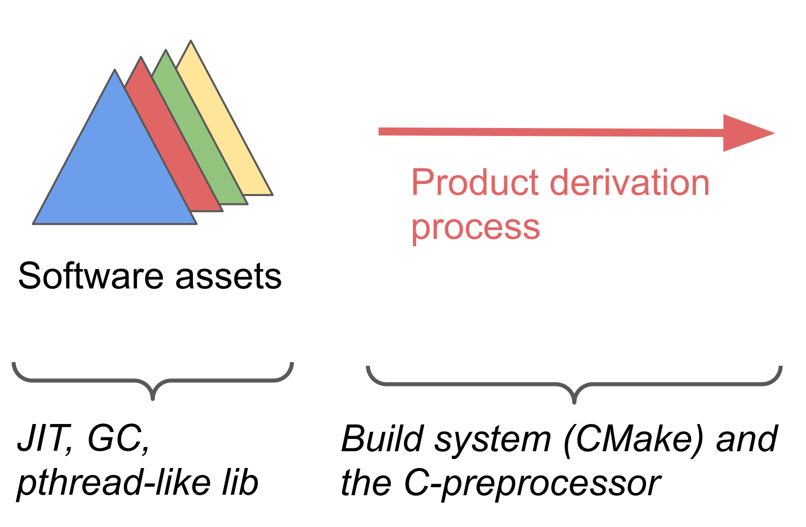


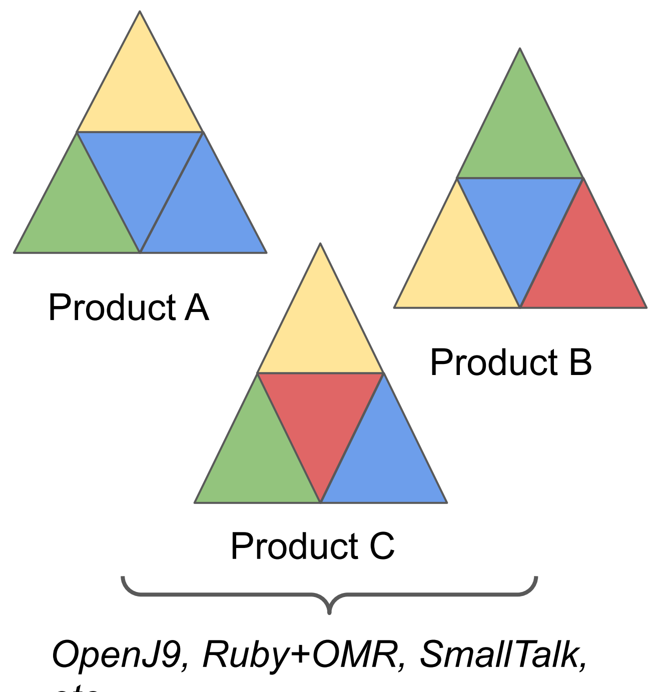
Examples include **MLPolyR** [1] and **Polyglot** [2].

References:
1. W. Chae and M. Blume, "Building a Family of Compilers," *SPLC '08*, Limerick, 2008.
2. Polyglot Extensible Compiler Framework, https://github.com/polyglot-compiler/polyglot.

Industrial case studies

In OMR, we are trying to re-engineer an already **existing highly-configurable system**.





Observation

Off-the-shelf mechanisms are not applicable for OMR.

Observation
Off-the-shelf mechanisms are not applicable for OMR.

Instead, tackle problems in an incremental manner.

Back to

C++ Enum/Union Extensibility Problem

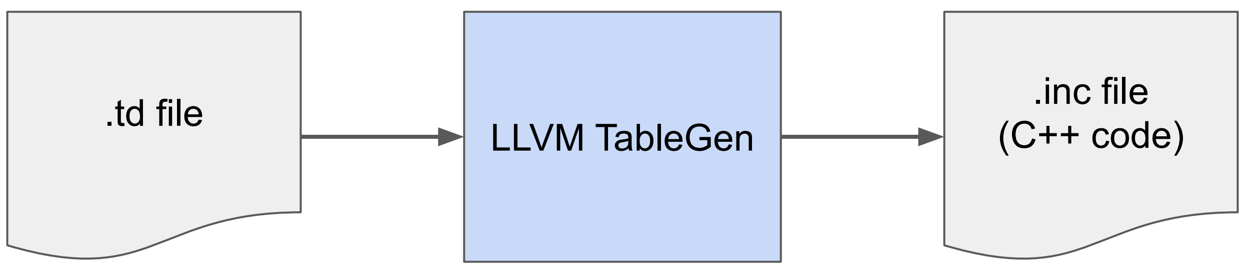
C++ enum/union extensibility problem
Potential solutions

Direction I:
Domain-specific language (DSL)

Direction II:
The C preprocessor and macros

References:
- https://github.com/eclipse/omr/issues/4519
- https://github.com/eclipse/omr/pull/4915
- https://youtu.be/21yPv8GsvY4

DSL Solution I: Custom DSL




```
def GR32 : RegisterClass<[i32], 32,
  [EAX, ECX, EDX, ESI, EDI, EBX, EBP, ESP,
   R8D, R9D, R10D, R11D, R14D, R15D, R12D, R13D]> { … }
```

Reference:

- https://llvm.org/docs/TableGen/
- https://www.aosabook.org/en/llvm.html

DSL Solution I: Custom DSL

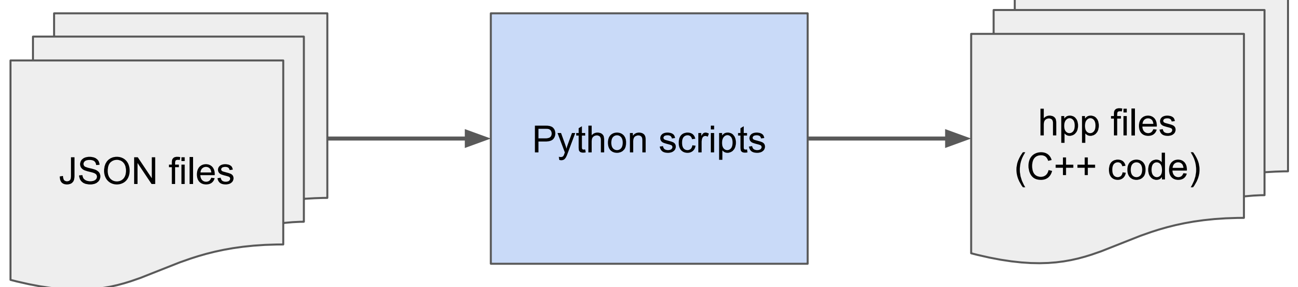
Pros

- Makes code more reusable
- Easier to track information (e.g., opcode props)

Cons

- Requires tool support (e.g., parser)
- OMR devs and clients will have to learn the DSL
- May add unnecessary deps

DSL Solution II: Python + JSON



DSL Solution II: Python + JSON

Pros

- No need to track C++ headers anymore
- Easier to change/extend opcodes and their props
- JSON is easy to understand and use

Cons

- Requires Python on multiple platforms as well as build servers
- Clients will have to have Python on their platform
- May add unnecessary deps

The C preprocessor (macro) based solution

```
#define FOR_EACH_OPCODE(MACRO)\
    // Opcode A
    MACRO(“iconst”, 1) \

    // Opcode B
    MACRO(“fconst”, 2) \

    ...
#endif
```

The C preprocessor (macro) based solution

```
#define FOR_EACH_OPCODE(MACRO)\
    // Opcode A
    MACRO("iconst", 1) \

    // Opcode B
    MACRO("fconst", 2) \

    ...
#endif
```

```
const char* names[] = {
#define GET_NAME(name, index) name,
    FOR_EACH_OPCODE(GET_NAME)
#undef GET_NAME
};
```

The C preprocessor (macro) based solution

```
#define FOR_EACH_OPCODE(MACRO)\
    // Opcode A
    MACRO("iconst", 1) \

    // Opcode B
    MACRO("fconst", 2) \

    ...
#endif
```

```
const char* names[] = {
#define GET_NAME(name, index) name,
    FOR_EACH_OPCODE(GET_NAME)
#undef GET_NAME
};
```

The C preprocessor (macro) based solution

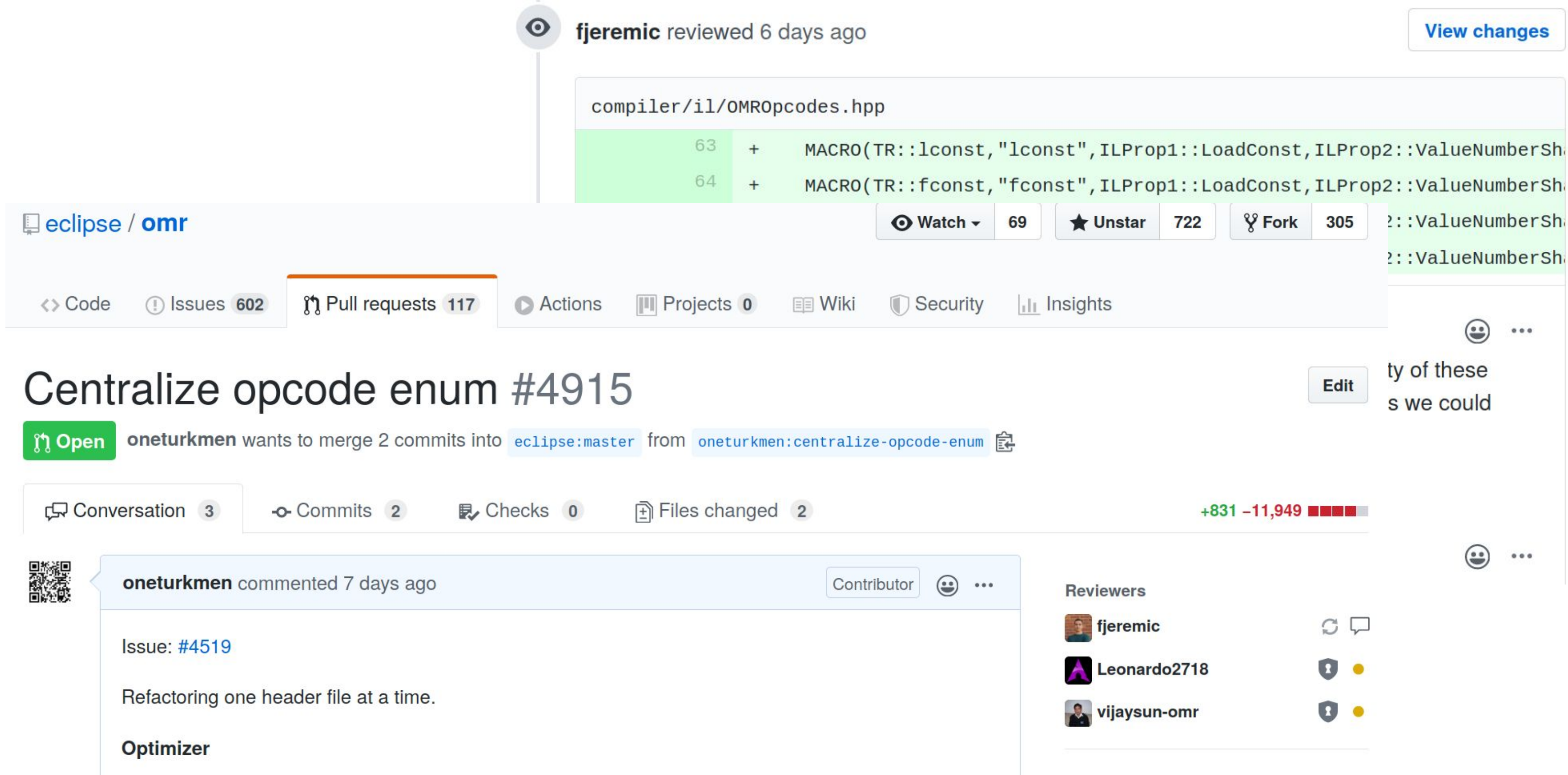
```
#define FOR_EACH_OPCODE(MACRO)\
    // Opcode A
    MACRO("iconst", 1) \

    // Opcode B
    MACRO("fconst", 2) \

    ...
#endif
```

```
const char* names[] = {
#define GET_NAME(name, index) name,
    FOR_EACH_OPCODE(GET_NAME)
#undef GET_NAME
};
```

==

```
const char* names[] = {
    "iconst",
    "fconst",
};
```

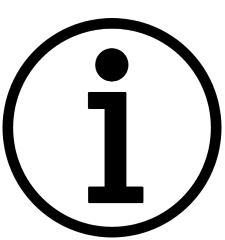
Current status of the solution



Current status of the solution

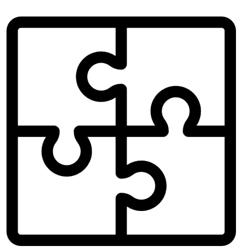
Header file containing **735 opcodes**, each containing **14 properties**.

Replace the old content of **12 header files** with **a single macro in each**.



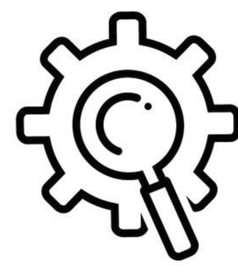
Icon made by Freepik from www.flaticon.com

Lessons learned



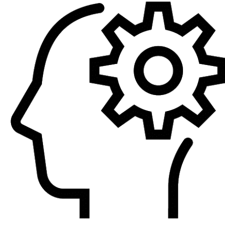
Understanding the bigger picture

There is more constraints and challenges we did not know about



Practical considerations

There is no one-fits-all solution



Large rehauls require expert knowledge

Deep knowledge of each piece of the code base is sometimes required

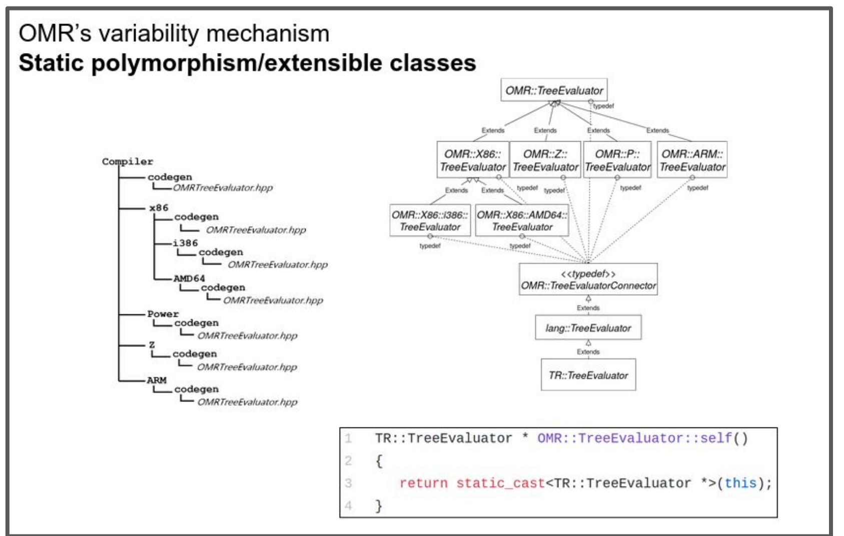
Puzzle icon made by Freepik from www.flaticon.com
Practical icon from ClipDealer
Expertise Icon #318416

Acknowledgements

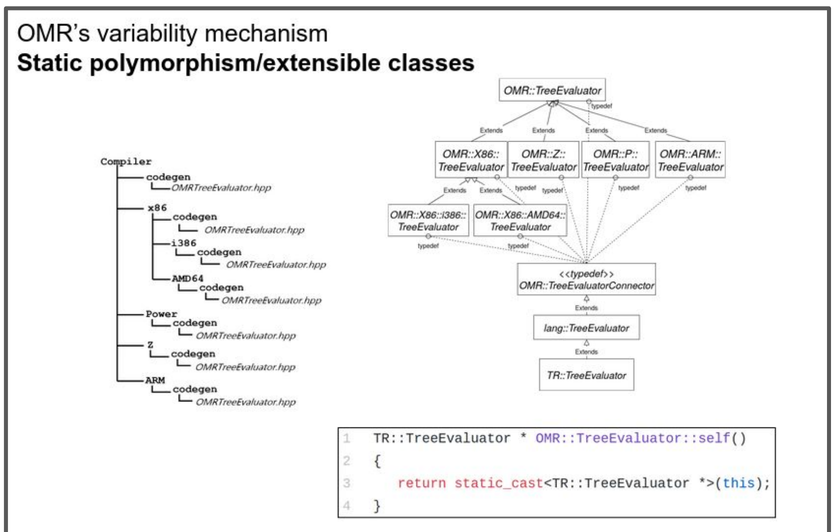
Special thanks go to:

- **Xiaoli Liang**
- **Nazim Bhuiyan**
- **Daryl Maier**

Summary



Summary



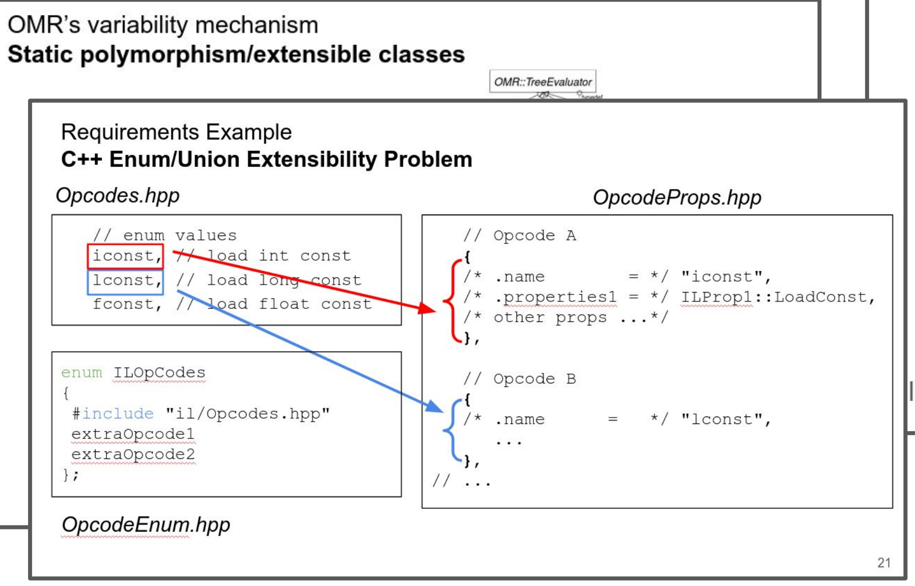

```
TR::TreeEvaluator * OMR::TreeEvaluator::self()
{
   return static_cast<TR::TreeEvaluator *>(this);
}
```

Challenges, constraints, and requirements

Simplicity and **usability**	Mechanism for **extending C++ enums/unions**	**No strongly connected components** (e.g., templates)
Enable **finer control** over extension points	More **streamlined consistency checks**	**Varying constructors** across archs/langs

All requirements are listed here: https://youtu.be/F7FIE1QIUAE

Summary

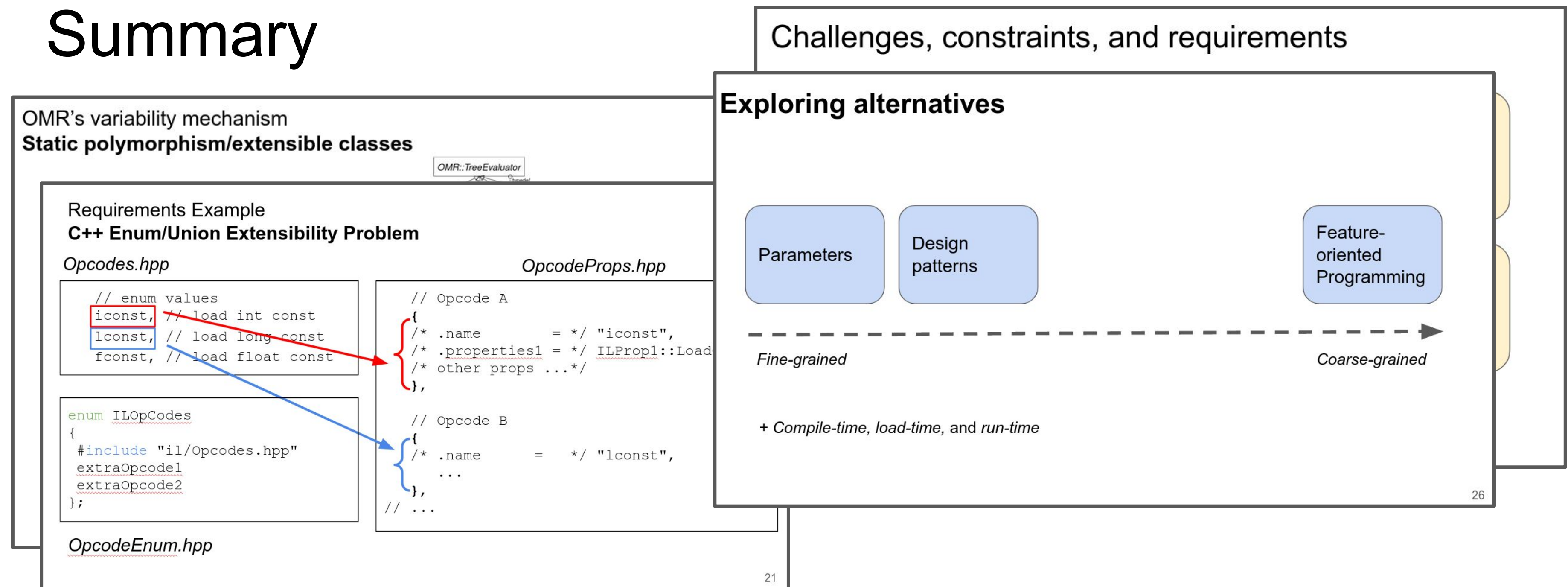


Challenges, constraints, and requirements

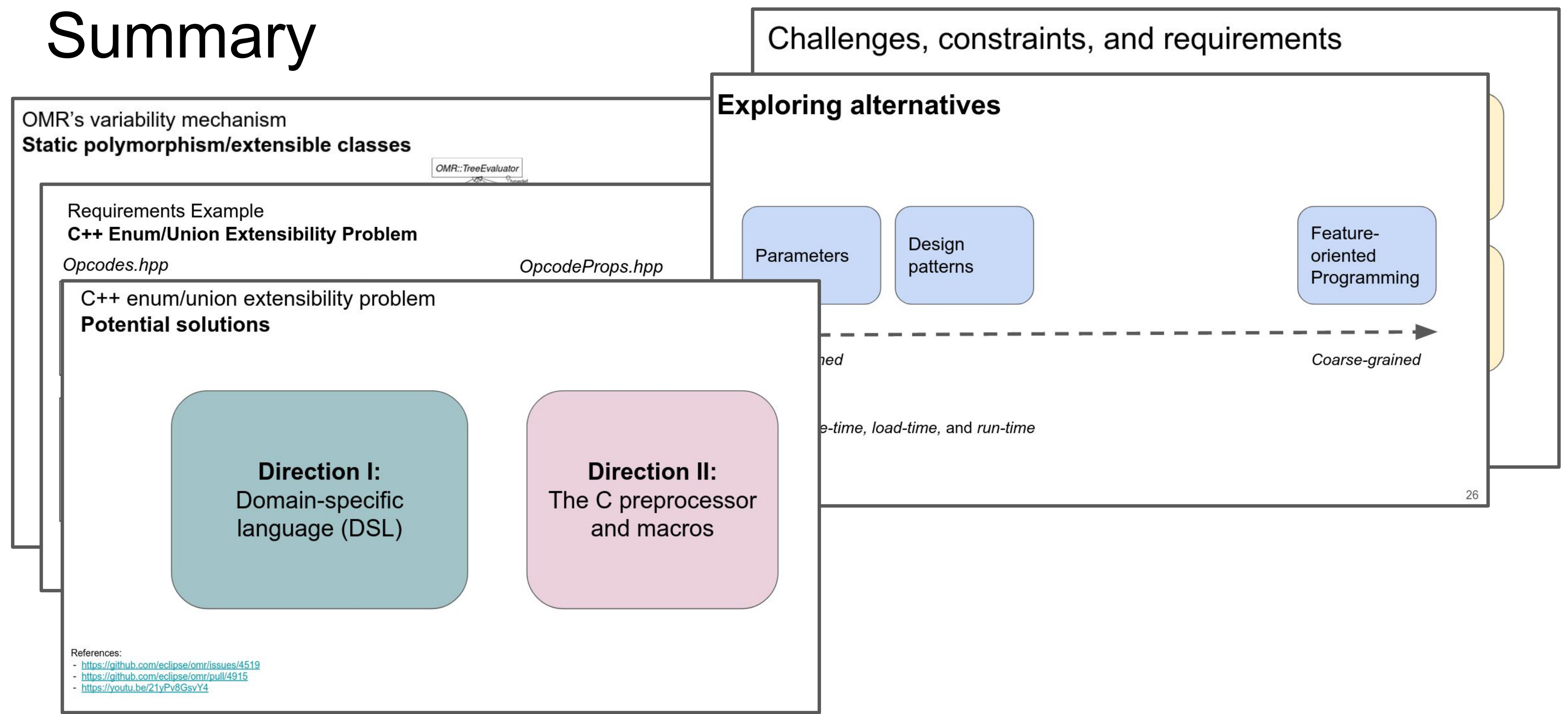
- **Simplicity** and **usability**
- Mechanism for **extending C++ enums/unions**
- **No strongly connected components** (e.g., templates)
- Enable **finer control** over extension points
- More **streamlined consistency checks**
- **Varying constructors** across archs/langs

l requirements are listed here: https://youtu.be/F7FIE1QIUAE

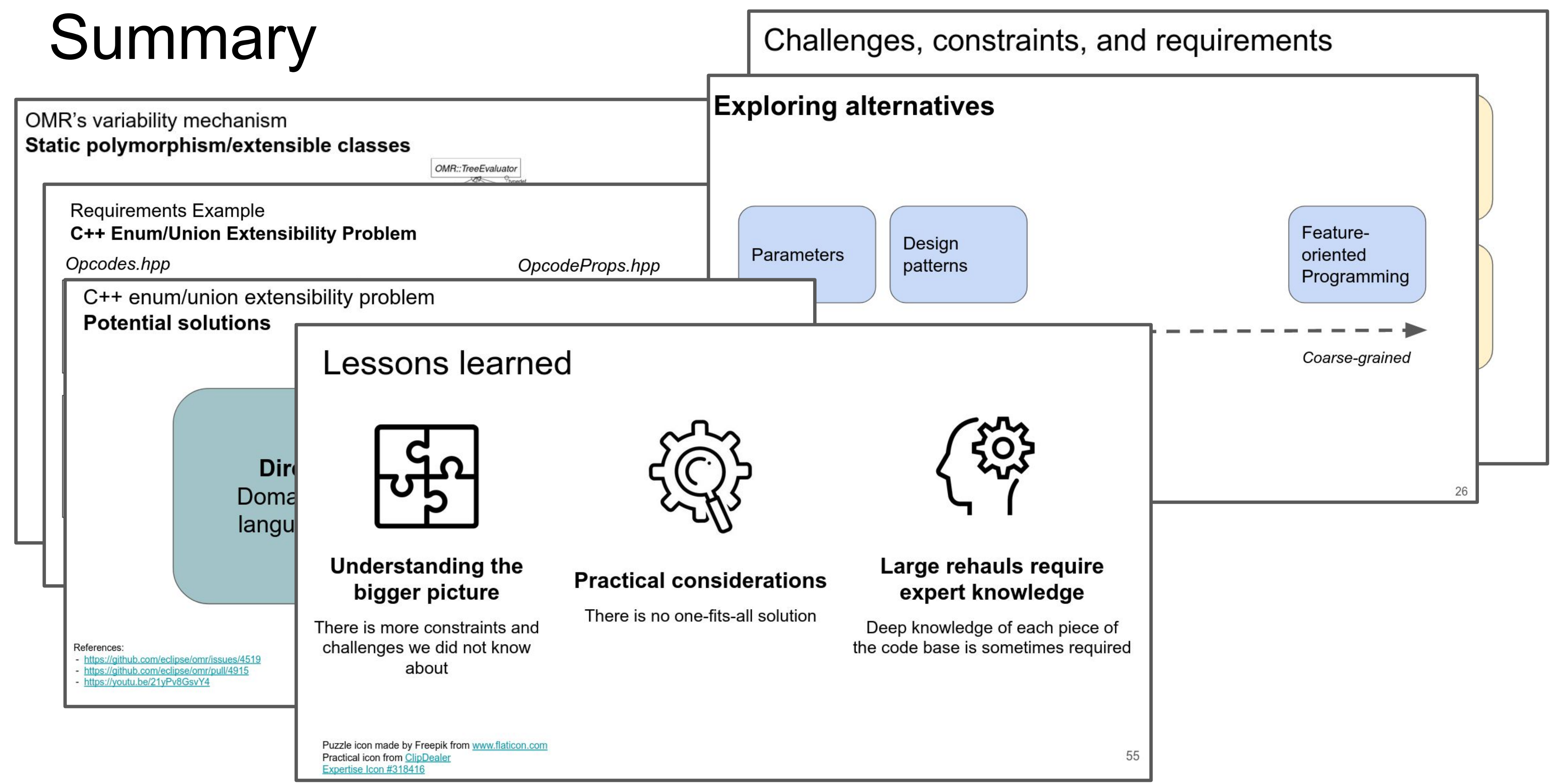
Summary



Summary



Summary



Summary