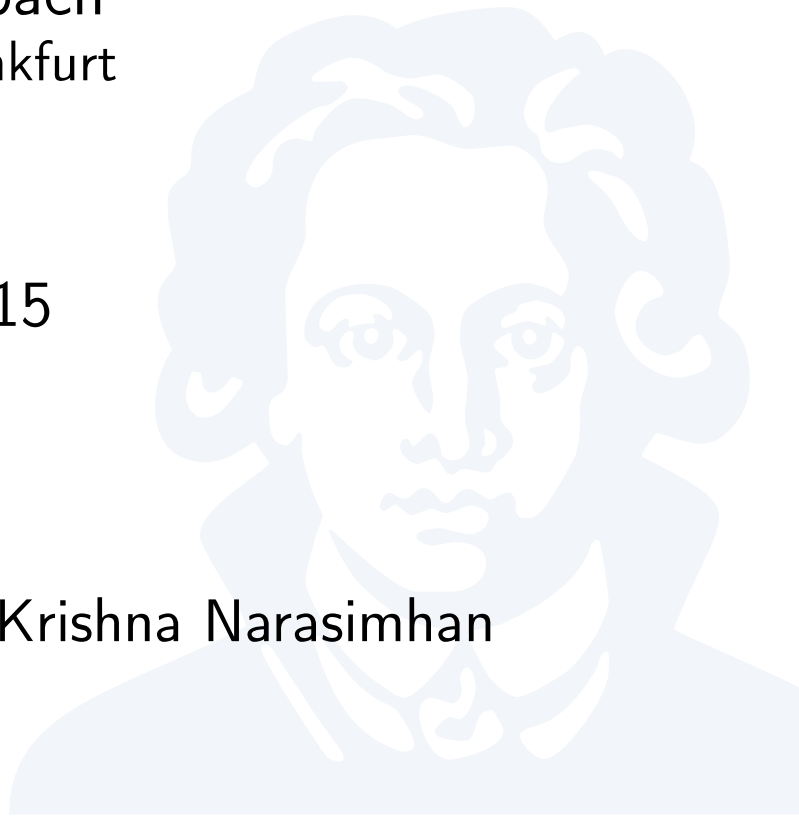
Copy-and-Paste Redeemed

Towards Adapting Abstractions

Christoph Reichenbach
Goethe University Frankfurt

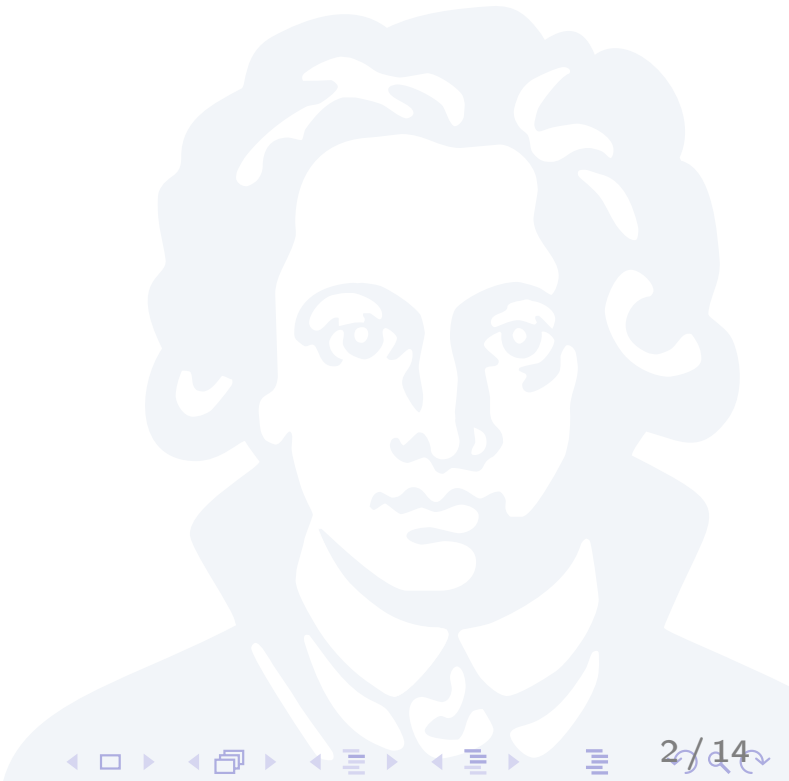
09 November 2015

based on our ASE 2015 paper with Krishna Narasimhan



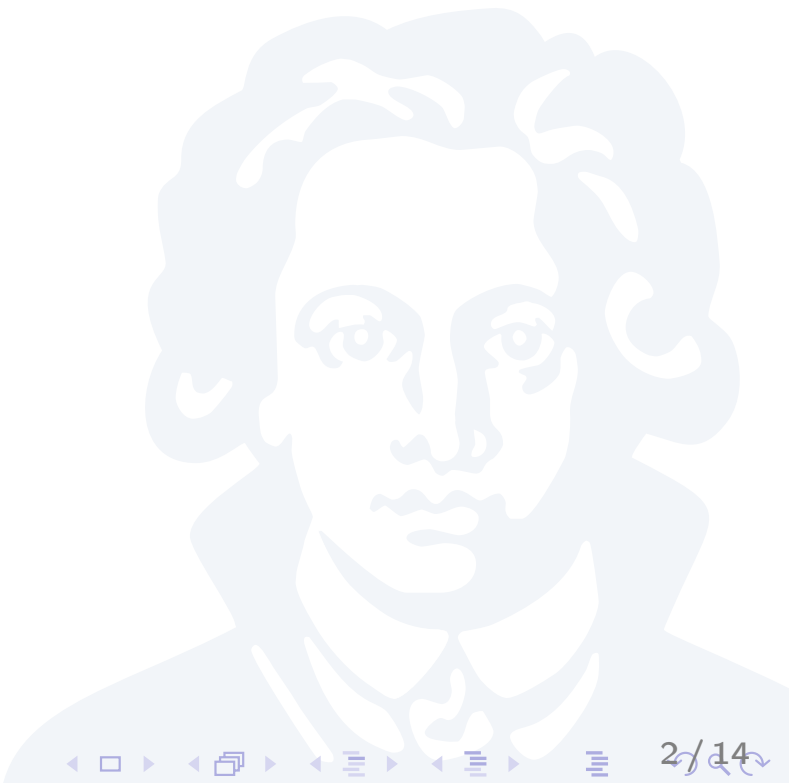
Extending DSLs

- **PQL/Java**: Parallel Query Language for Java
- **PQL-ESL**: Extension Specification Language for PQL/Java



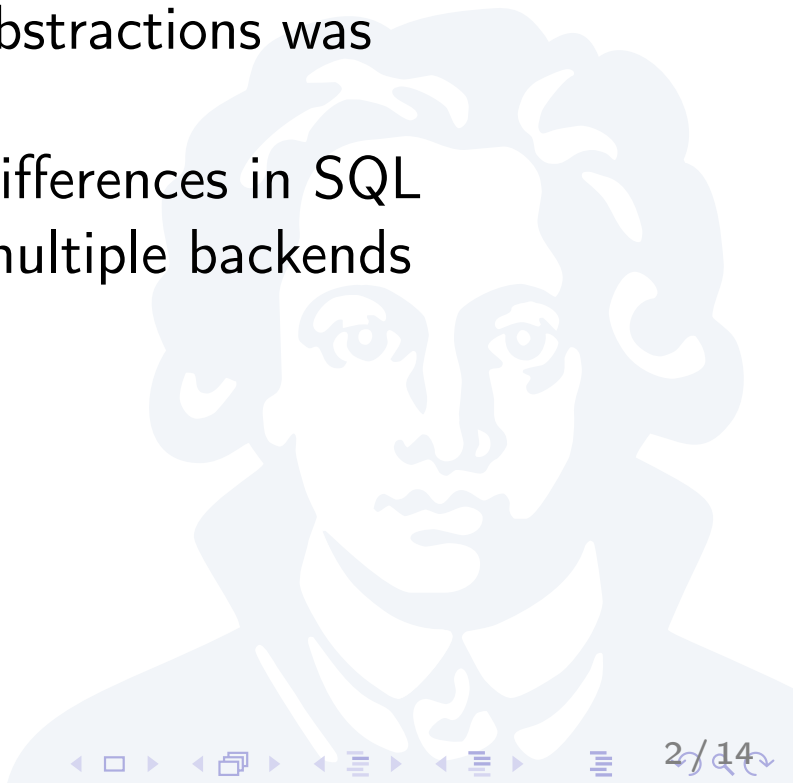
Extending DSLs

- **PQL/Java**: Parallel Query Language for Java
- **PQL-ESL**: Extension Specification Language for PQL/Java
- Student Project: Add SQL support via PQL-ESL



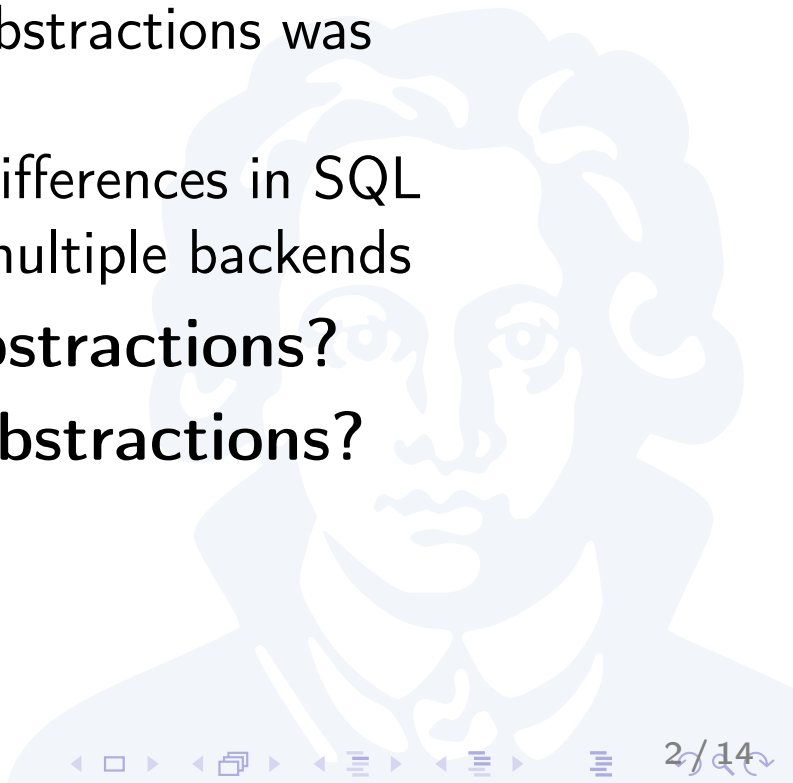
Extending DSLs

- **PQL/Java**: Parallel Query Language for Java
- **PQL-ESL**: Extension Specification Language for PQL/Java
- Student Project: Add SQL support via PQL-ESL
  - BSc student: "Using PQL-ESL abstractions was challenging"
  - SQL support still not optimal: Differences in SQL implementations would require multiple backends



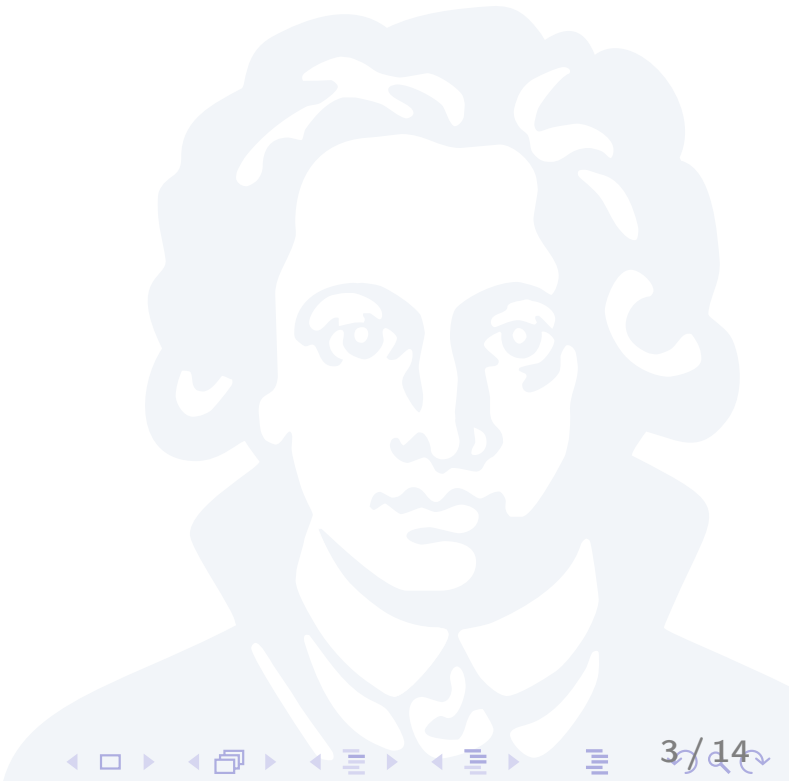
Extending DSLs

- **PQL/Java**: Parallel Query Language for Java
- **PQL-ESL**: Extension Specification Language for PQL/Java
- Student Project: Add SQL support via PQL-ESL
  - BSc student: "Using PQL-ESL abstractions was challenging"
  - SQL support still not optimal: Differences in SQL implementations would require multiple backends
- **Can we help introduce new abstractions?**
- **Can we help extend existing abstractions?**



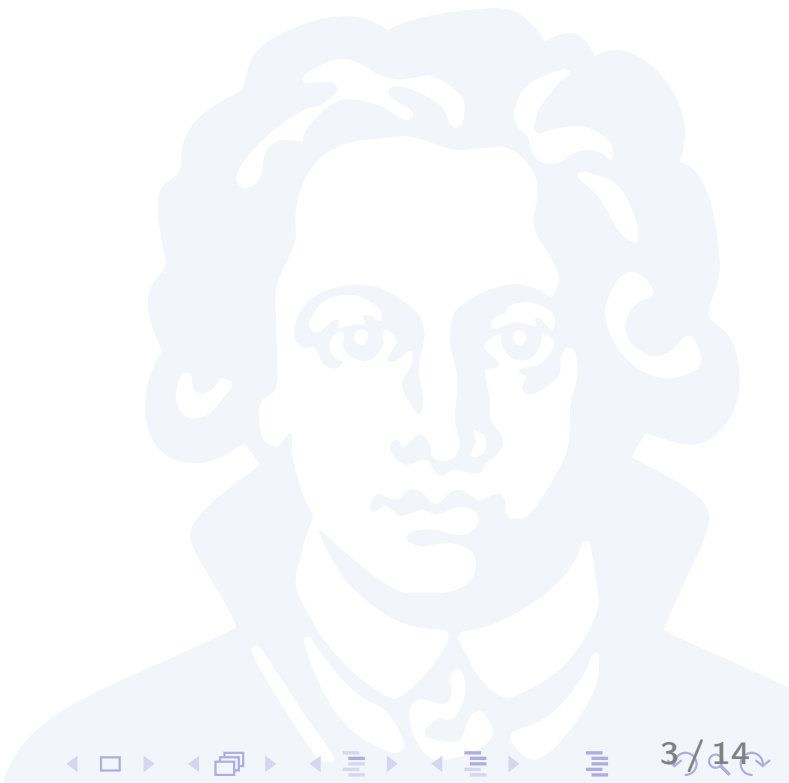
Duplication vs. Abstraction

```
int dist(coord a, coord b)
{
  int dx = abs(a.x - b.x);
  int dy = abs(a.y - b.y);
  return std::max(dx, dy);
}
```



Duplication vs. Abstraction

```
int dist(coord a, coord b)
{
  int dx = abs(a.x - b.x);
  int dy = abs(a.y - b.y);
  return std::max(dx, dy);
}
int dist2(coord a, coord b)
{
  int dx = abs(a.x - b.x);
  int dy = abs(a.y - b.y);
  return std::hypot(dx, dy);
}
```

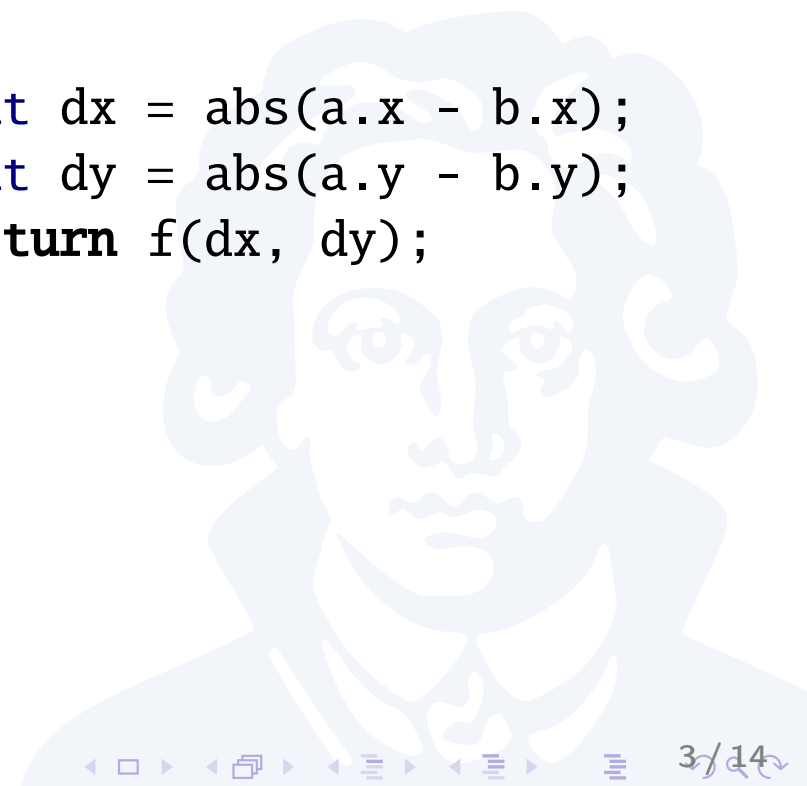


Duplication vs. Abstraction

```
int dist(coord a, coord b)
{
  int dx = abs(a.x - b.x);
  int dy = abs(a.y - b.y);
  return std::max(dx, dy);
}
int dist2(coord a, coord b)
{
  int dx = abs(a.x - b.x);
  int dy = abs(a.y - b.y);
  return std::hypot(dx, dy);
}
```

Alternative: Abstraction

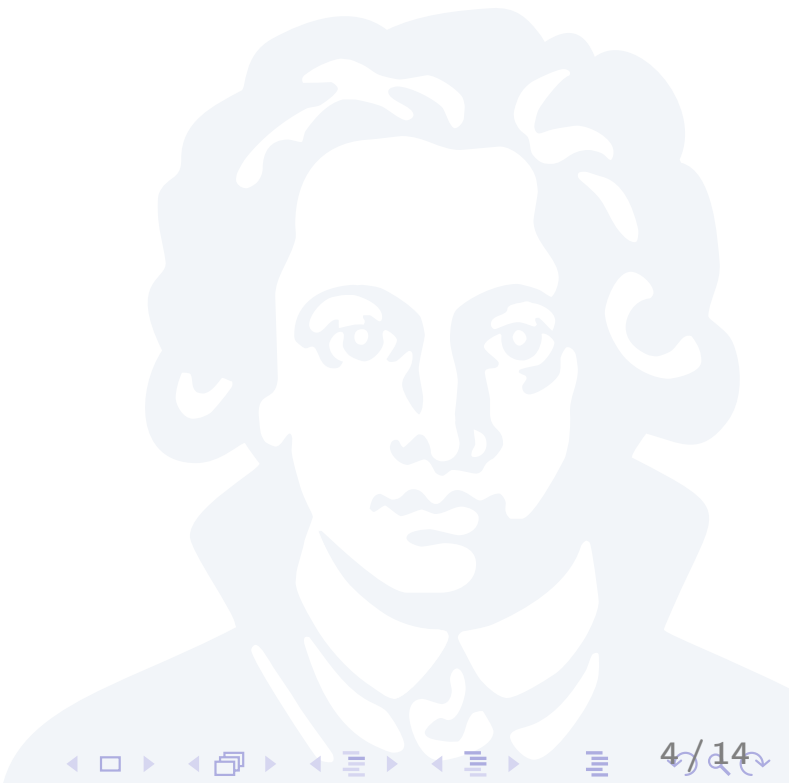
```
int distG(coord a, coord b,
          int (*f)(int, int))
{
  int dx = abs(a.x - b.x);
  int dy = abs(a.y - b.y);
  return f(dx, dy);
}
```



A Small User Study

- We took a number of near-clones from existing programs
- We removed one of the clones from each program
- We asked five C++ programmers (graduate students) to reimplement missing functionality
  - Experience: 2, 3, 12, 48, 120 months
- **Abstraction**:

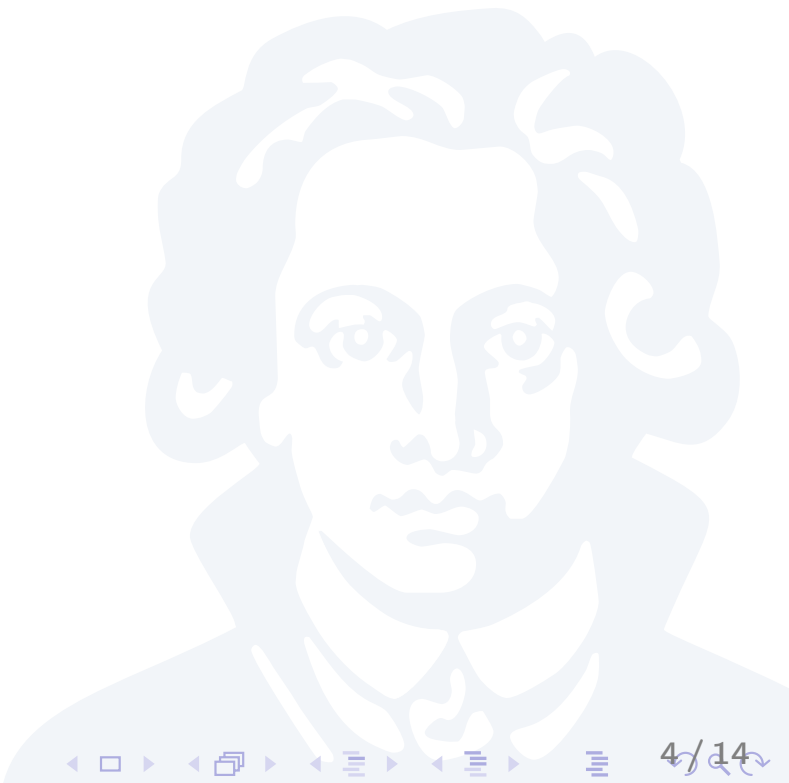
- **Copy-Paste**:



A Small User Study

- We took a number of near-clones from existing programs
- We removed one of the clones from each program
- We asked five C++ programmers (graduate students) to reimplement missing functionality
  - Experience: 2, 3, 12, 48, 120 months
- **Abstraction**:
  - Users tend to prefer abstraction

- **Copy-Paste**:



A Small User Study

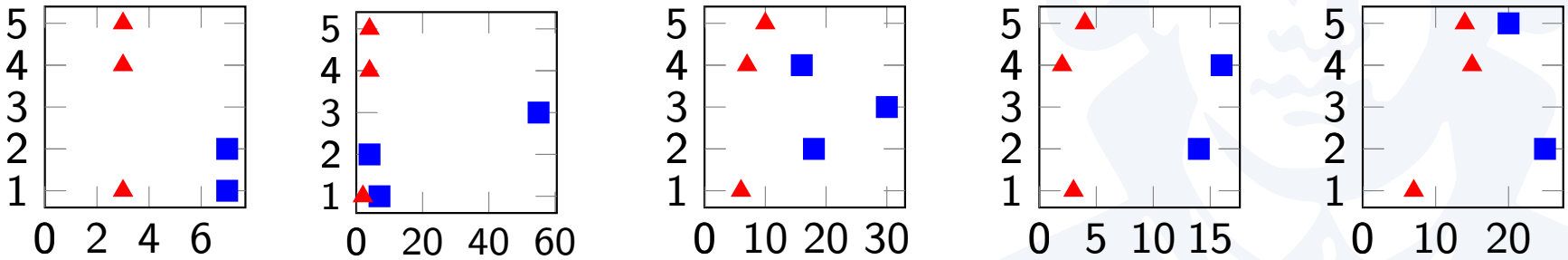
- We took a number of near-clones from existing programs
- We removed one of the clones from each program
- We asked five C++ programmers (graduate students) to reimplement missing functionality
  - Experience: 2, 3, 12, 48, 120 months
- **Abstraction**:
  - Users tend to prefer abstraction
  - 7/10 tasks completed in time
- **Copy-Paste**:
  - 15/15 tasks completed in time

A Small User Study

- We took a number of near-clones from existing programs
- We removed one of the clones from each program
- We asked five C++ programmers (graduate students) to reimplement missing functionality
  - Experience: 2, 3, 12, 48, 120 months
- **Abstraction**:
  - Users tend to prefer abstraction
  - 12/15 tasks completed in time
- **Copy-Paste**:
  - 15/15 tasks completed in time

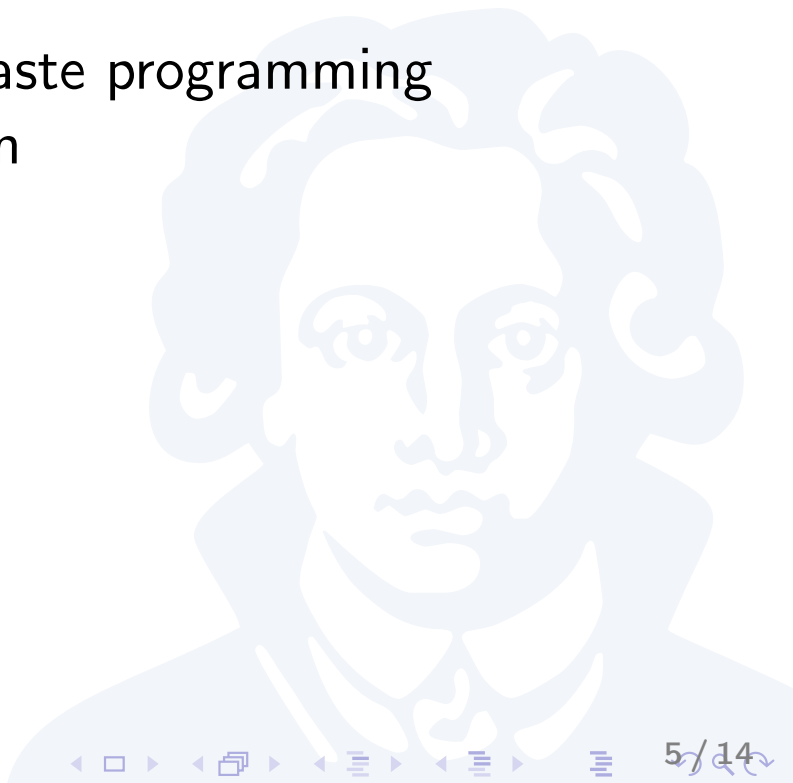
A Small User Study

- We took a number of near-clones from existing programs
- We removed one of the clones from each program
- We asked five C++ programmers (graduate students) to reimplement missing functionality
  - Experience: 2, 3, 12, 48, 120 months
- **Abstraction**: ■
  - Users tend to prefer abstraction
  - 12/15 tasks completed in time
  - Factor 2-3 slower than abstraction
- **Copy-Paste**: ▲
  - 15/15 tasks completed in time
  - Universally faster (except for a single tie)



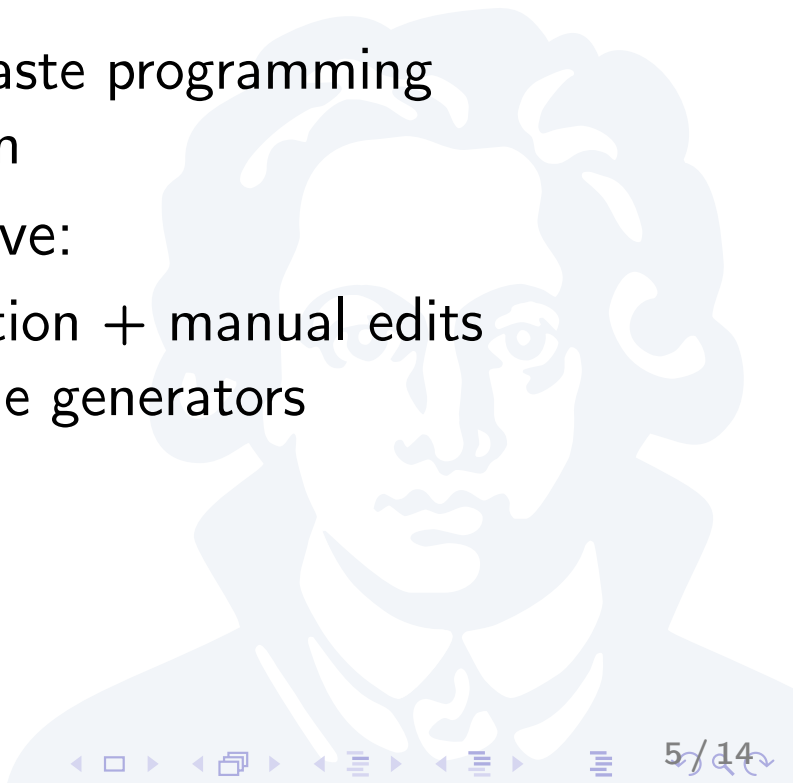
The Best of Both Worlds

- Hypothesis: More abstractions $\Rightarrow$ slower programmers
- However:
  - Literature on clone research mostly agrees that clones are maintenance problem
  - Practitioner literature strongly advocates abstraction
- Choice between:
  - **Fast development** with copy-paste programming
  - **Maintainability** with abstraction



The Best of Both Worlds

- Hypothesis: More abstractions $\Rightarrow$ slower programmers
- However:
  - Literature on clone research mostly agrees that clones are maintenance problem
  - Practitioner literature strongly advocates abstraction
- Choice between:
  - **Fast development** with copy-paste programming
  - **Maintainability** with abstraction
- Generative Programming perspective:
  - **Fast development** with generation + manual edits
  - **Maintainability** by evolving code generators



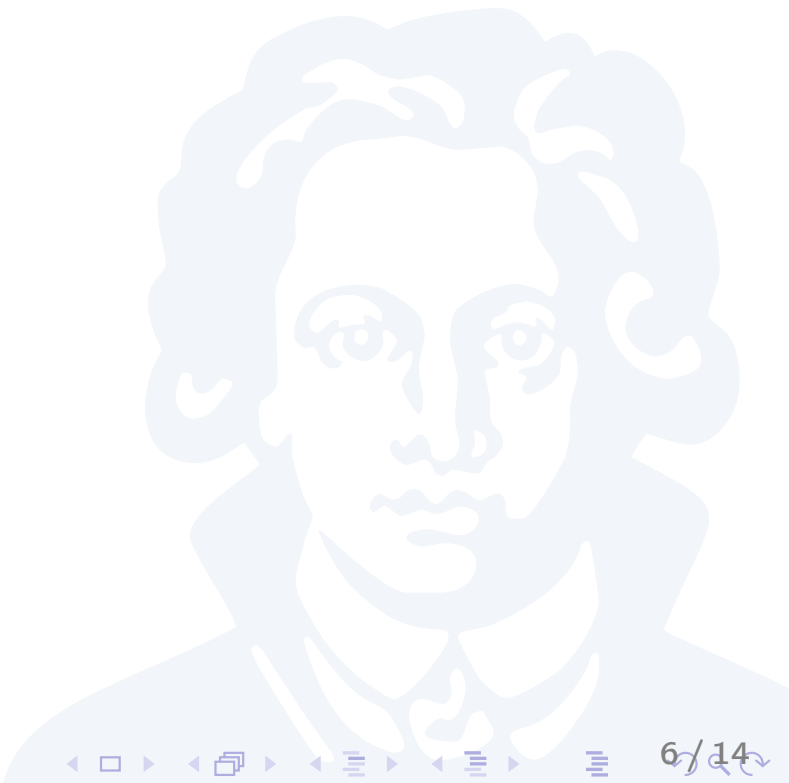
The Best of Both Worlds

- Hypothesis: More abstractions $\Rightarrow$ slower programmers
- However:
  - Literature on clone research mostly agrees that clones are maintenance problem
  - Practitioner literature strongly advocates abstraction
- Choice between:
  - **Fast development** with copy-paste programming
  - **Maintainability** with abstraction
- Generative Programming perspective:
  - **Fast development** with generation + manual edits
  - **Maintainability** by evolving code generators

Why not both?

An Example

```
void function1()
{
  fake_db_init(&fakedb, true);
  run_mode = MODE_DEBUG;
  start(&service);
}
void function2()
{
  fake_db_init(&fakedb, false);
  run_mode = MODE_INFO;
  start(&service);
}
void function3()
{
  db_init();
  db_sanity_check();
  run_mode = MODE_REGULAR;
  start(&service);
}
```

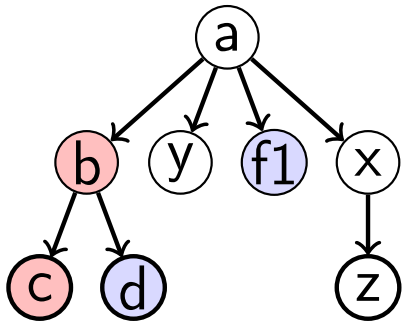


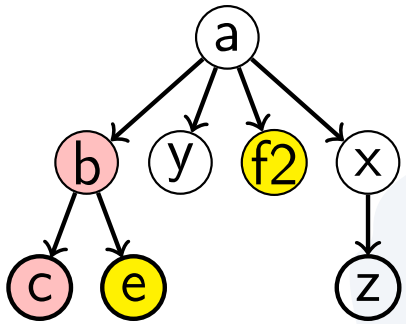
An Example

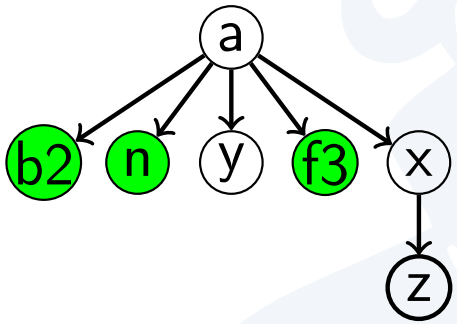
```
void function1()
{
  b(c,d);
  y = f1;
  x(z);
}
void function2()
{
  b(c,e);
  y = f2;
  x(z);
}
void function3()
{
  b2();
  n();
  y = f3;
  x(z);
}
```

An Example

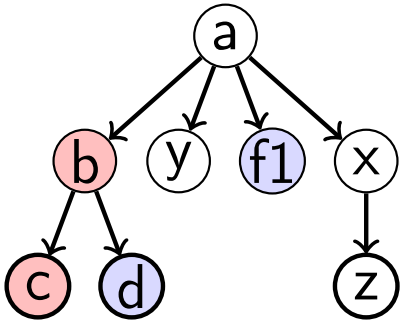
```
void function1()
{
  b(c,d);
  y = f1;
  x(z);
}
void function2()
{
  b(c,e);
  y = f2;
  x(z);
}
void function3()
{
  b2();
  n();
  y = f3;
  x(z);
}
```

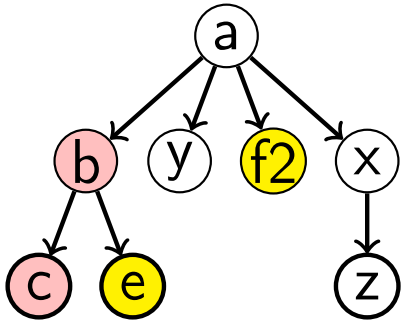


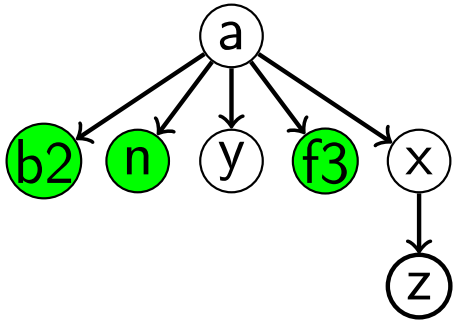




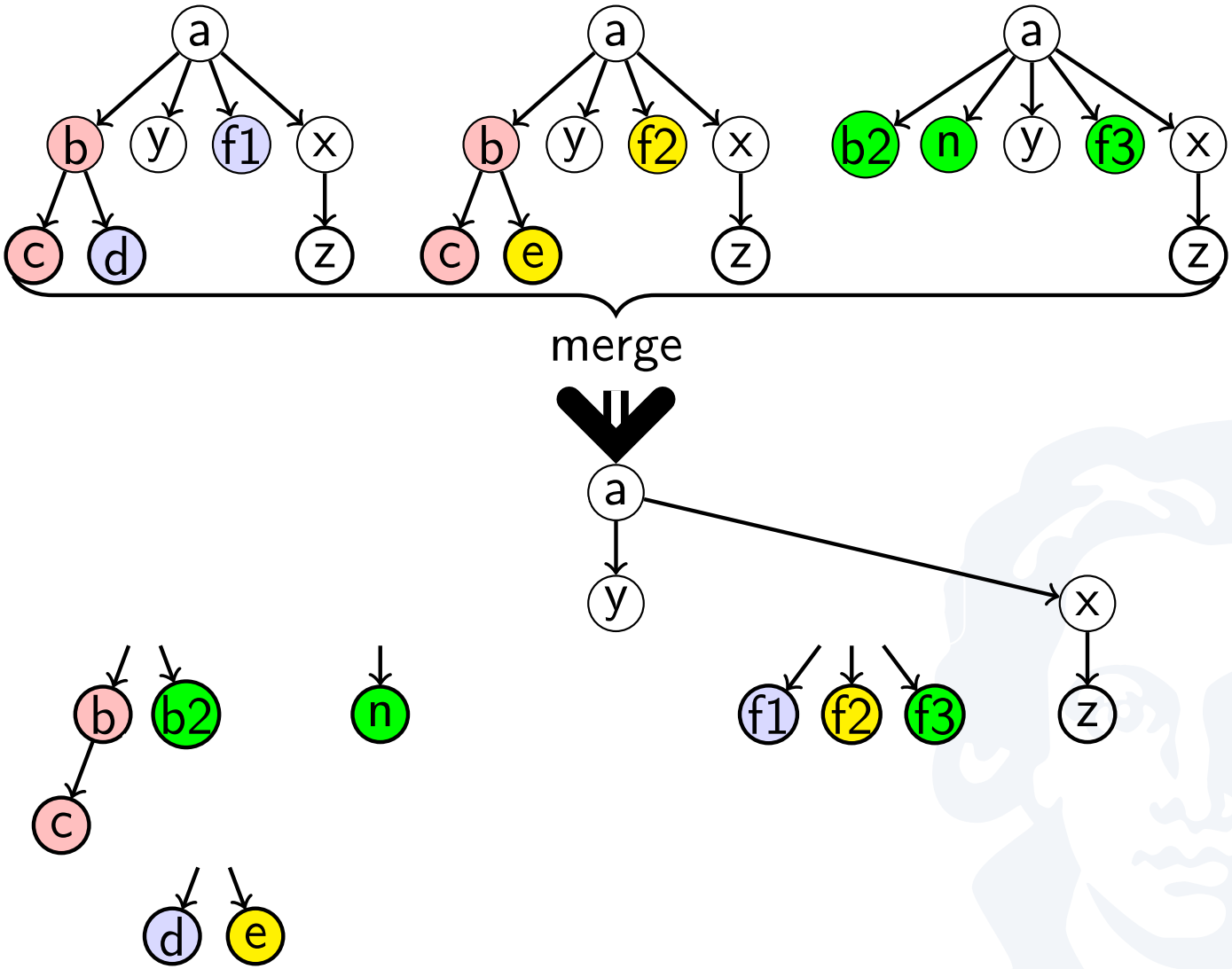
Automating Abstraction



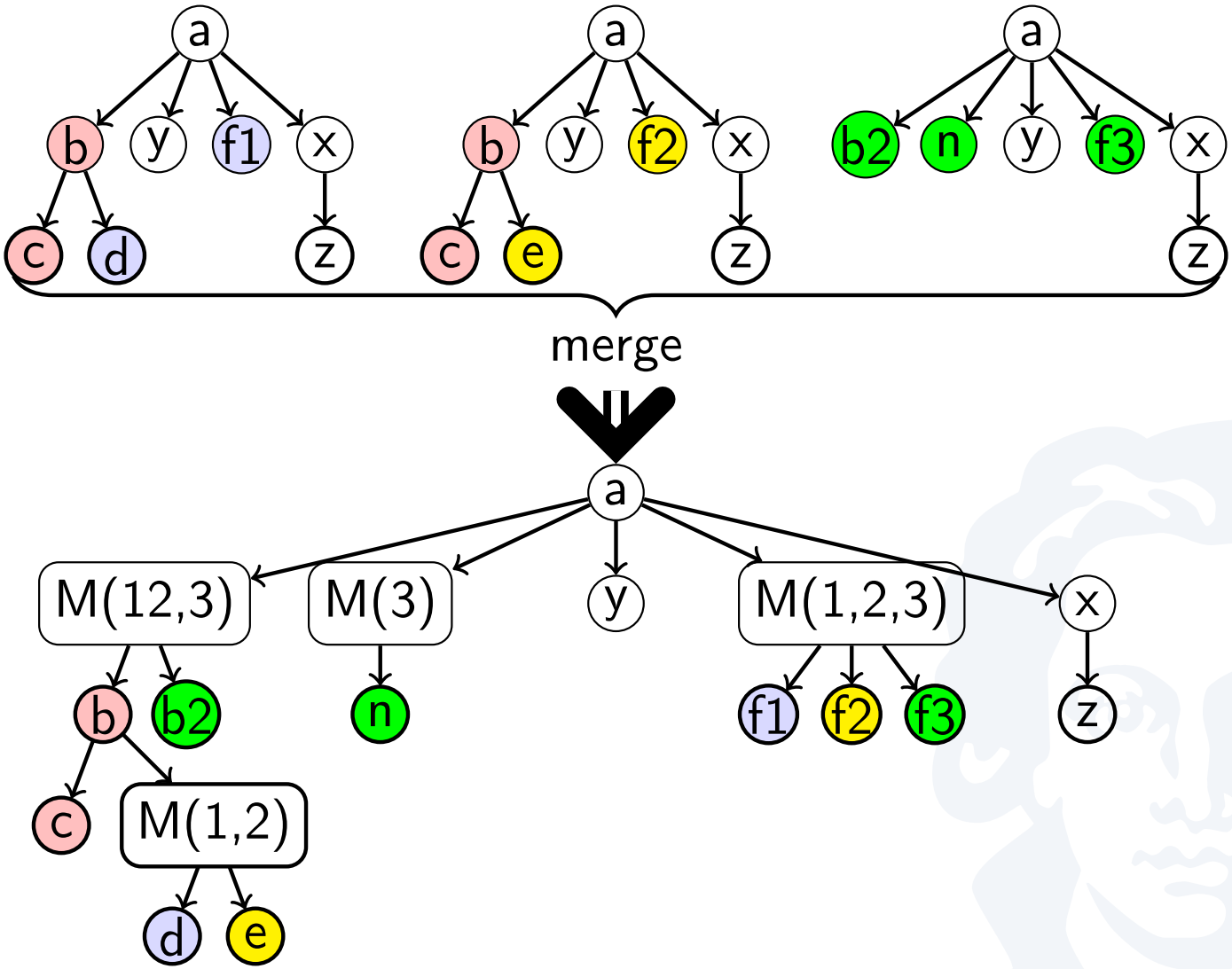




Automating Abstraction



Automating Abstraction



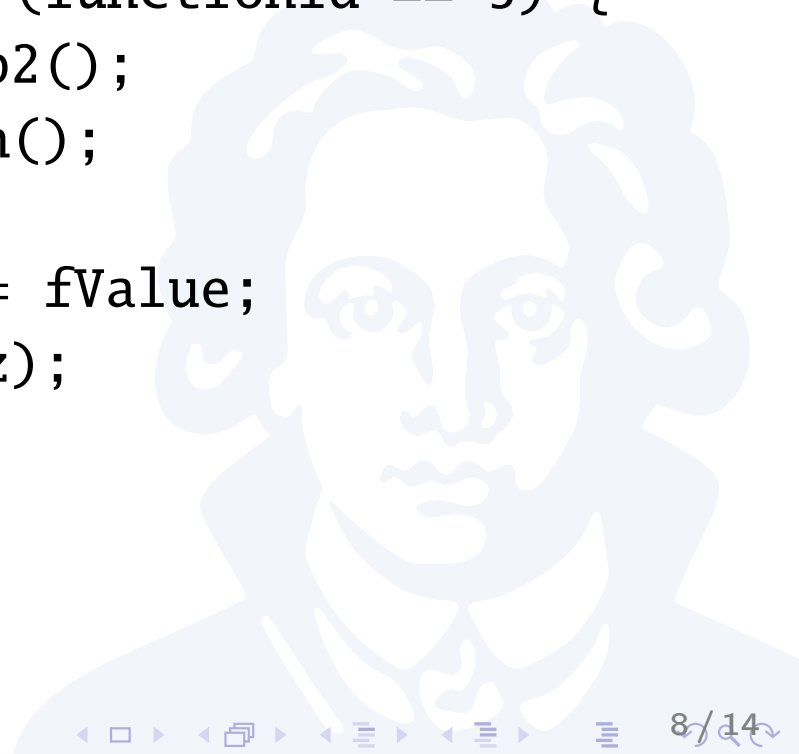
An Example

```
void function1()
{
  b(c,d);
  y = f1;
  x(z);
}
void function2()
{
  b(c,e);
  y = f2;
  x(z);
}
void function3()
{
  b2();
  n();
  y = f3;
  x(z);
}
```

An Example

```
void function1()
{
  b(c,d);
  y = f1;
  x(z);
}
void function2()
{
  b(c,e);
  y = f2;
  x(z);
}
void function3()
{
  b2();
  n();
  y = f3;
  x(z);
}
```

```
void fnMerged(int functionId,
     int fValue, bool bParam)
{
  if (functionId == 1
   || functionId == 2) {
    b(c, bParam);
  }
  if (functionId == 3) {
    b2();
    n();
  }
  y = fValue;
  x(z);
}
```



An Example

```
void function1()
{
  b(c,d);
  y = f1;
  x(z);
}
void function2()
{
  b(c,e);
  y = f2;
  x(z);
}
void function3()
{
  b2();
  n();
  y = f3;
  x(z);
}
```

```
void fnMerged(int functionId,
     int fValue, bool bParam)
{
  if (functionId == 1
   || functionId == 2) {
    b(c, bParam);
  }
  if (functionId == 3) {
    b2();
    n();
  }
  y = fValue;
  x(z);
}
void function1() {
  fnMerged(1,f1,d);
}
```

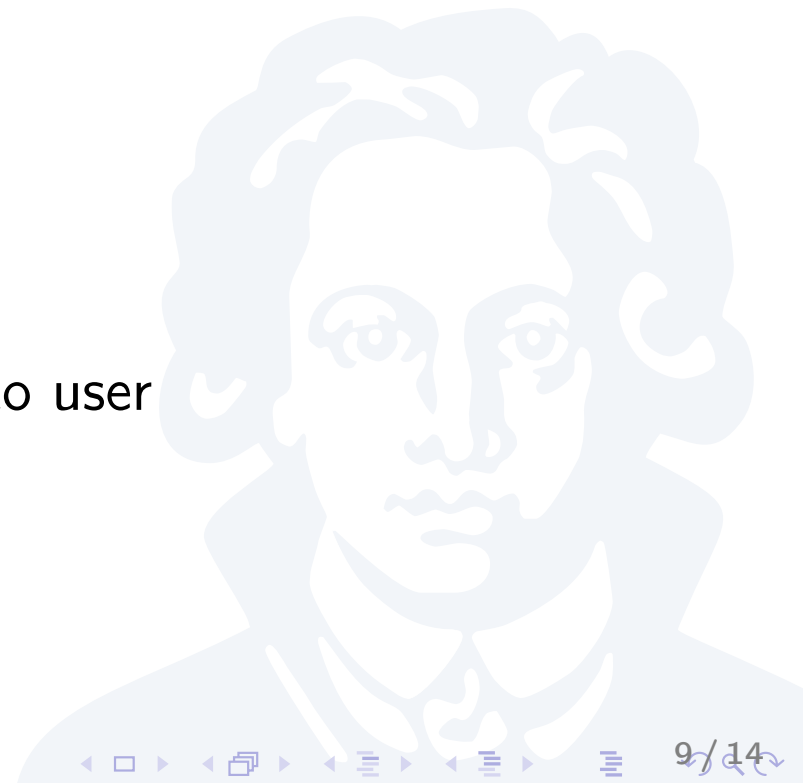
Realising Merge Points

- Merge Points (M) realised via abstraction
- Different forms of abstraction:
  - *C++*:
    - Parameters
    - Template parameters
    - Fields
    - Delegate patterns
    - Subclasses
    - Function pointers

    ...

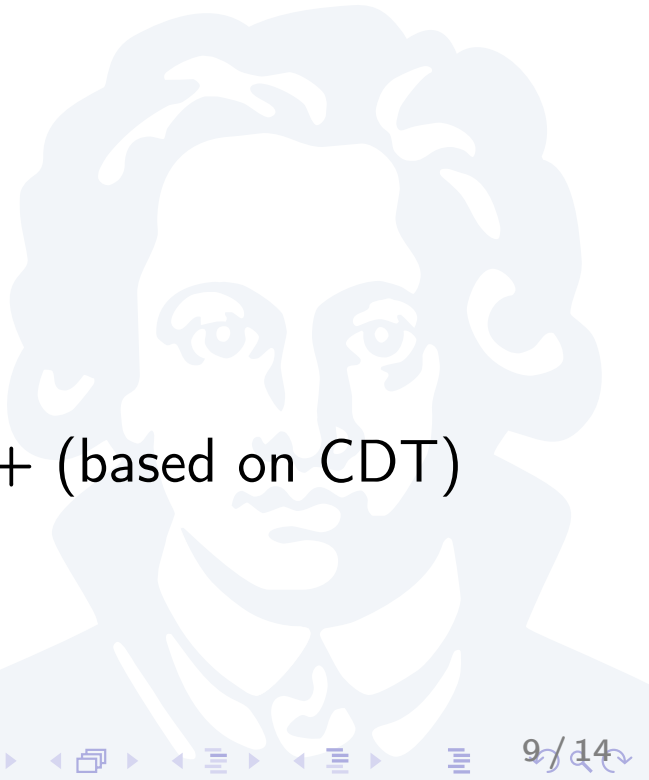
Realising Merge Points

- Merge Points (M) realised via abstraction
- Different forms of abstraction:
  - *C++*:
    - Parameters
    - Template parameters
    - Fields
    - Delegate patterns
    - Subclasses
    - Function pointers
    - ...
  - We leave choice of abstraction to user



Realising Merge Points

- Merge Points (M) realised via abstraction
- Different forms of abstraction:
  - *C++*:
    - **Parameters**
    - **Template parameters**
    - Fields
    - Delegate patterns
    - Subclasses
    - **Function pointers**
    - ...
  - We leave choice of abstraction to user
  - Implemented as Eclipse plugin for C++ (based on CDT)



Implementing Merges

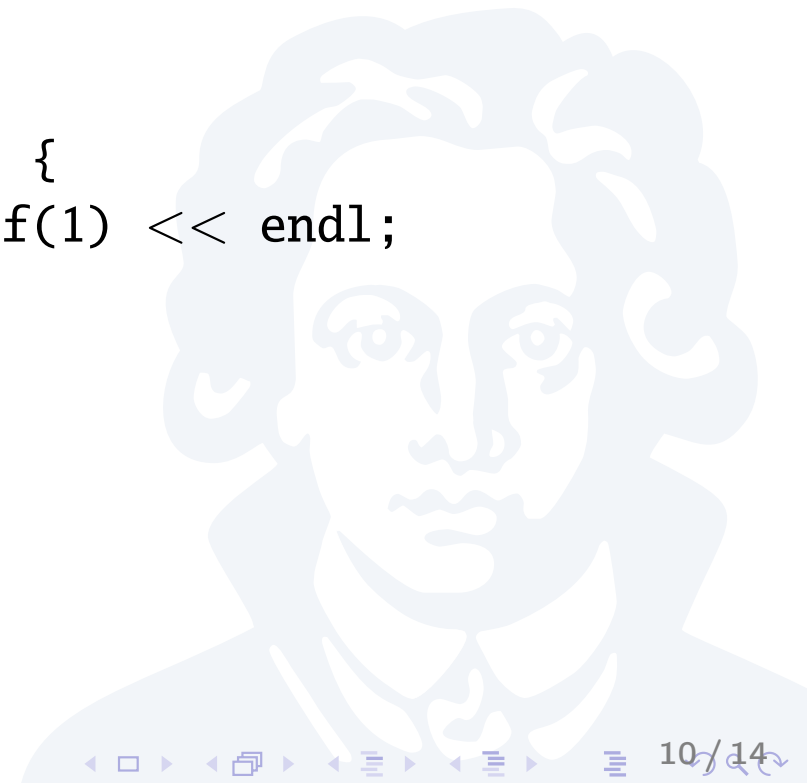
```
int f1() {
  return 1;
}

int f2() {
  return 2;
}

void main() {
  cout << f1() << endl;
}
```

```
int f(int k) {
  switch (k) {
  case 1: return 1;
  case 2: return 2;
  }
}

void main() {
  cout << f(1) << endl;
}
```

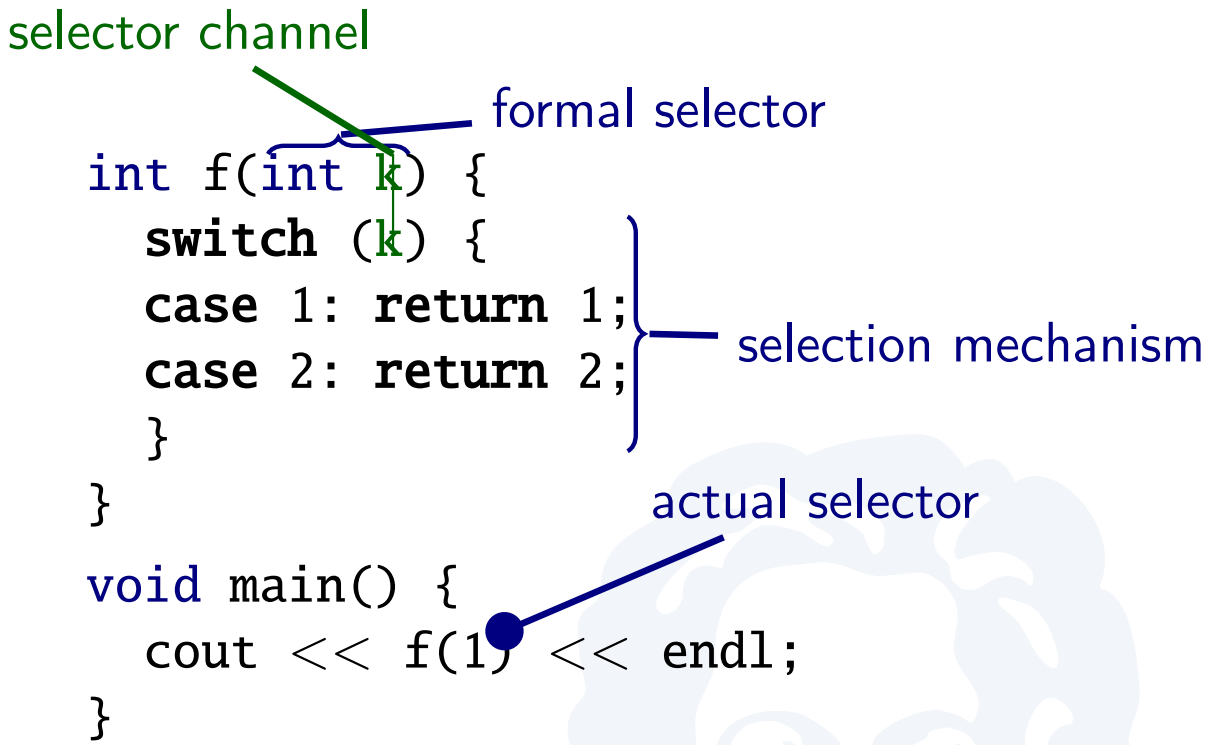


Implementing Merges

```
int f1() {
  return 1;
}

int f2() {
  return 2;
}

void main() {
  cout << f1() << endl;
}
```



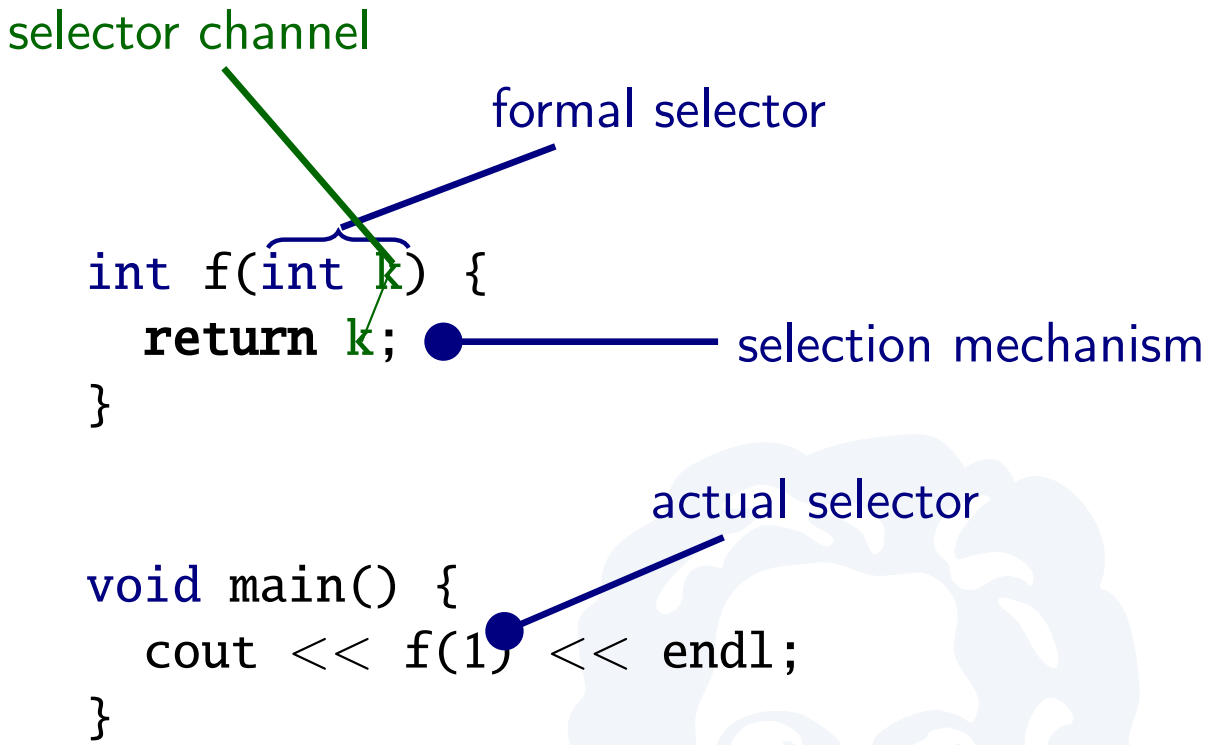
- **Selector**: encodes the variant to choose
- **Selection Mechanism**: translates selector to alternative

Implementing Merges

```
int f1() {
  return 1;
}

int f2() {
  return 2;
}

void main() {
  cout << f1() << endl;
}
```



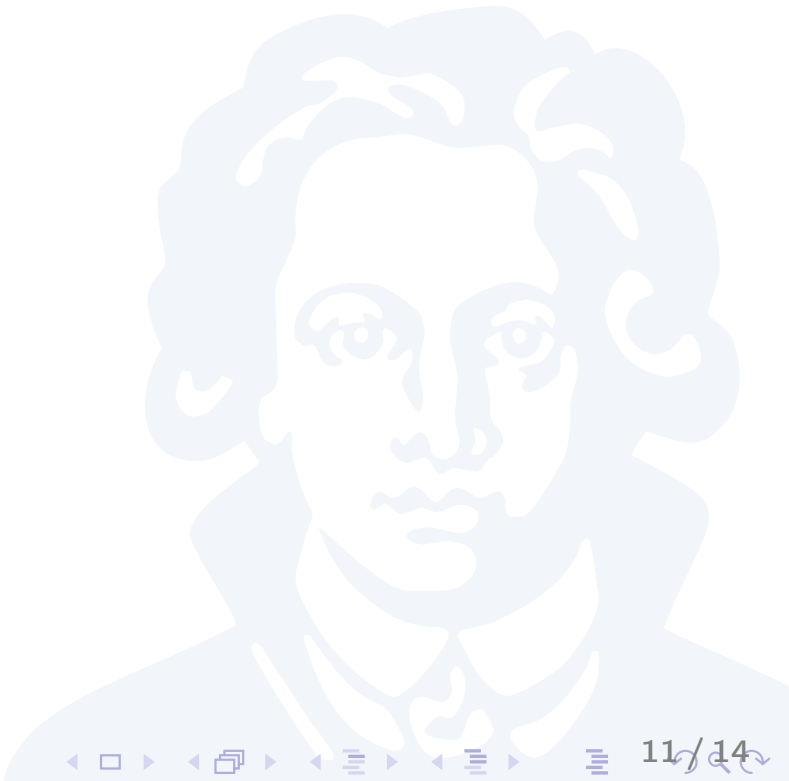
- **Selector**: encodes the variant to choose
- **Selection Mechanism**: translates selector to alternative

Selectors and Selection Mechanisms in C++

Selector Channel Types

- Value Parameter
- Pointer Parameter
- Reference Parameter
- Template Parameter
- Global Variable
- Field
- Dynamic Type
- UNIX Environment Variable

...



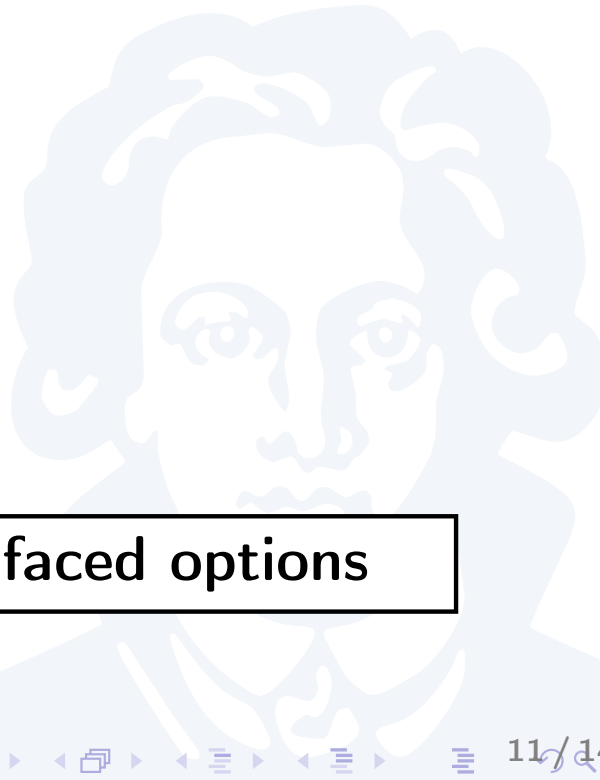
Selectors and Selection Mechanisms in C++

Selector Channel Types

- **Value Parameter**
- **Pointer Parameter**
- Reference Parameter
- **Template Parameter**
- Global Variable
- Field
- Dynamic Type
- UNIX Environment Variable

...

Our C++ prototype supports the boldfaced options



Selectors and Selection Mechanisms in C++

Selector Channel Types

- **Value Parameter**
- **Pointer Parameter**
- Reference Parameter
- **Template Parameter**
- Global Variable
- Field
- Dynamic Type
- UNIX Environment Variable

...

Selection Mechanism Types

- Direct Parameter
- *Switch* Statement
- *If* Statement
- Function Pointer
- Closure
- Delegates
- Dynamic Dispatch
- Dynamic Linker

...

Our C++ prototype supports the boldfaced options

Selectors and Selection Mechanisms in C++

Selector Channel Types

- **Value Parameter**
- **Pointer Parameter**
- Reference Parameter
- **Template Parameter**
- Global Variable
- Field
- Dynamic Type
- UNIX Environment Variable

...

Selection Mechanism Types

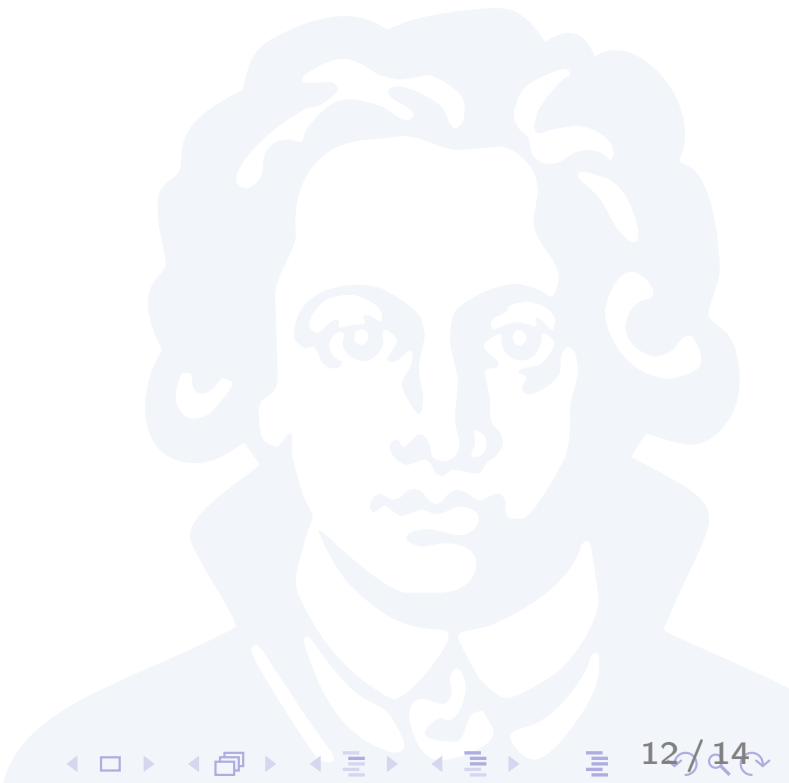
- **Direct Parameter**
- ***Switch* Statement**
- ***If* Statement**
- **Function Pointer (*partial*)**
- Closure
- Delegates
- Dynamic Dispatch
- Dynamic Linker

...

Our C++ prototype supports the boldfaced options

Evaluation

- Selected near-clones from Open Source C++ projects (Facebook rocksdb, Google protobuf, Oracle node-oracledb, mongodb, ...)
- Used our tool to merge
- Manual cleanup: Formatting, variable renaming

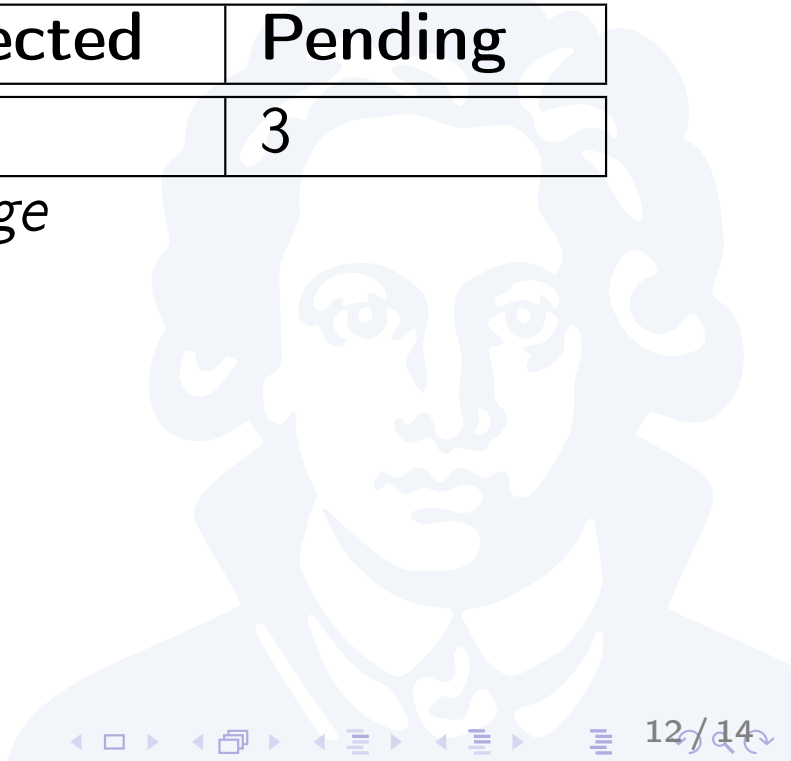


Evaluation

- Selected near-clones from Open Source C++ projects (Facebook rocksdb, Google protobuf, Oracle node-oracledb, mongodb, ...)
- Used our tool to merge
- Manual cleanup: Formatting, variable renaming
- Phase 1 (Early prototype, limited to two-way merge):

Submitted	**Accepted**	**Rejected**	**Pending**
8	1	4	3

$\Rightarrow$ Feedback: *Need multi-way merge*



Evaluation

- Selected near-clones from Open Source C++ projects (Facebook rocksdb, Google protobuf, Oracle node-oracledb, mongodb, ...)
- Used our tool to merge
- Manual cleanup: Formatting, variable renaming
- Phase 1 (Early prototype, limited to two-way merge):

Submitted	Accepted	Rejected	Pending
8	1	4	3

  $\Rightarrow$ Feedback: *Need multi-way merge*

- Phase 2:

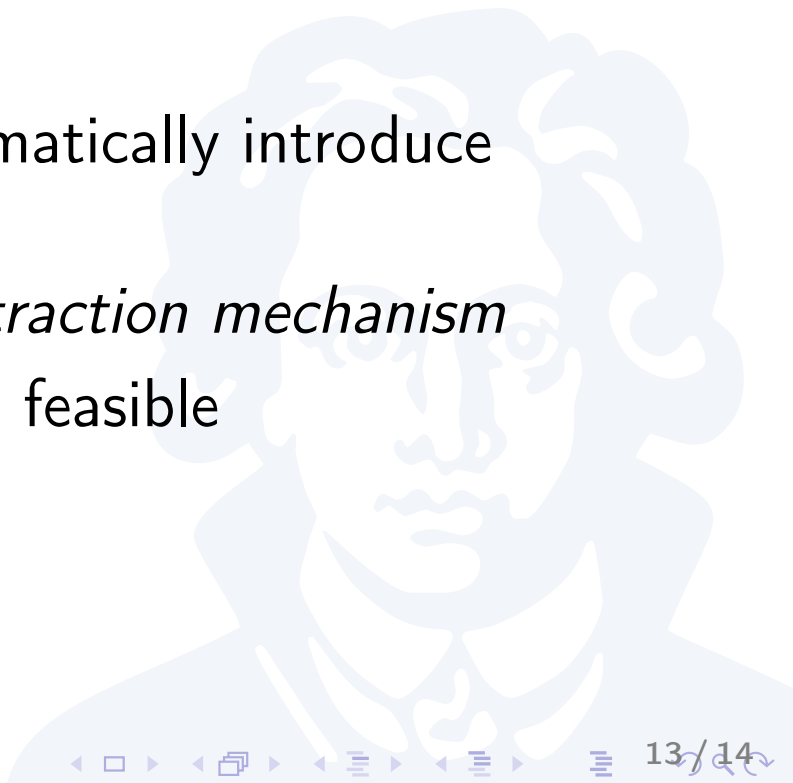
Submitted	Accepted	Rejected	Pending
10	9	0	1

Effective as a Refactoring Tool

Lessons for Generative Programming?

Language-Level Abstraction

- Approach shows that we can automatically introduce abstraction
- Still need heuristics to choose *abstraction mechanism*
- Extending existing abstraction also feasible

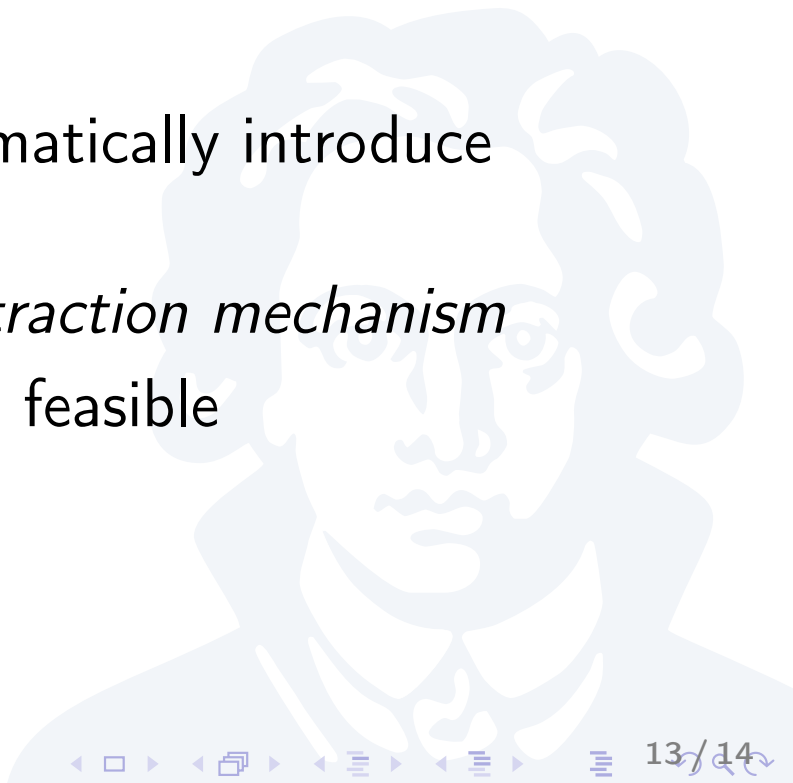


Lessons for Generative Programming?

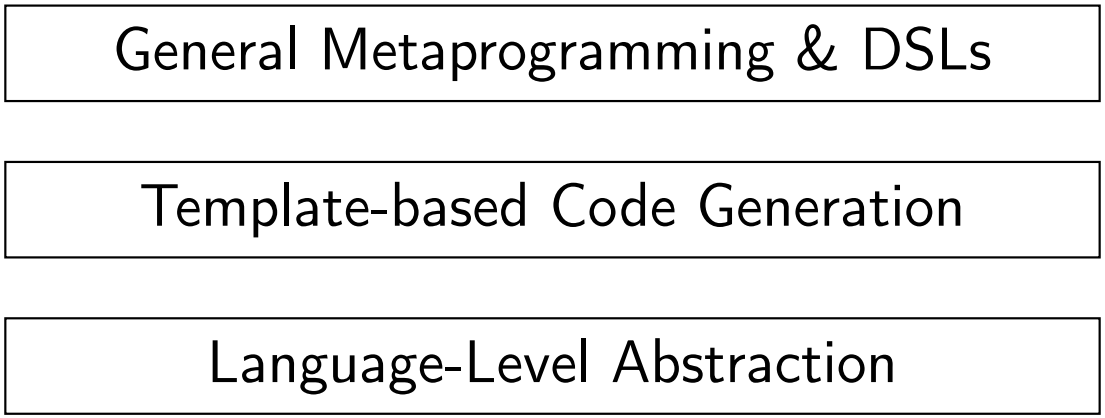
Template-based Code Generation

Language-Level Abstraction

- Approach shows that we can automatically introduce abstraction
- Still need heuristics to choose *abstraction mechanism*
- Extending existing abstraction also feasible

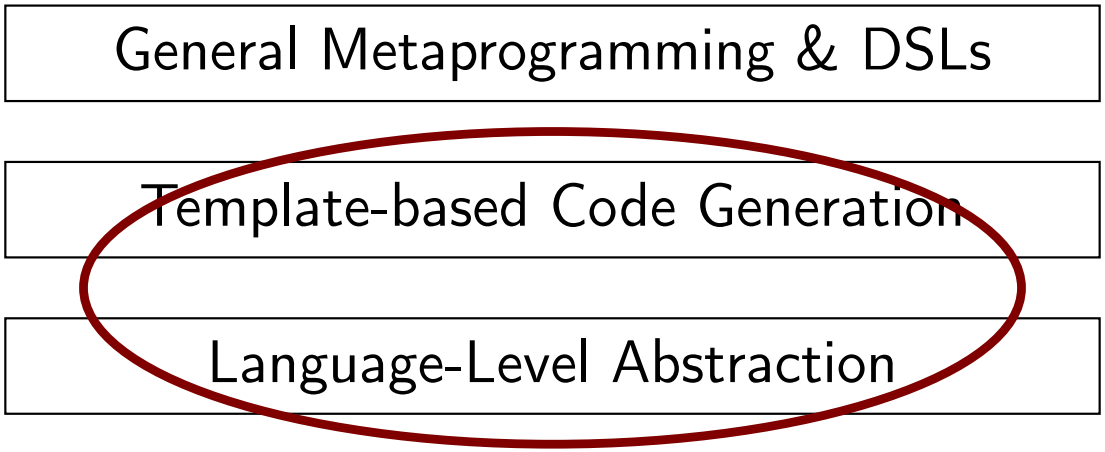


Lessons for Generative Programming?



- Approach shows that we can automatically introduce abstraction
- Still need heuristics to choose *abstraction mechanism*
- Extending existing abstraction also feasible

Lessons for Generative Programming?



- Approach shows that we can automatically introduce abstraction
- Still need heuristics to choose *abstraction mechanism*
- Extending existing abstraction also feasible

Open Question: How do we pick the Actual Selector automatically?

Conclusions

- Hypothesis: More abstractions $\Rightarrow$ harder for humans to work with
- Idea: Use automation to
  - Abstract code after cloning
  - Specialise abstract code prior to *specialised* modification (*Inline* refactoring)
  - Merge modification back into abstract code
- For C++/Eclipse: `https://marketplace.eclipse.org/content/clone-abstractor-c-methods-0`