Clang Based Refactoring

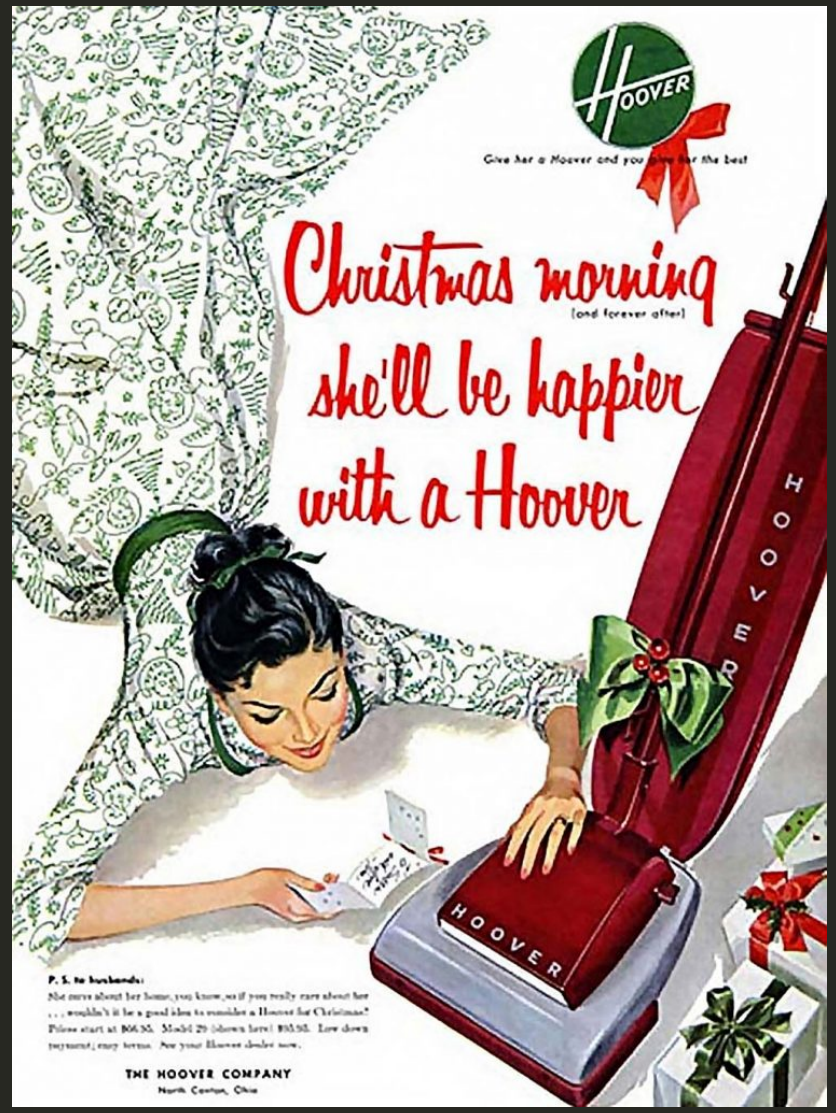
How to refactor millions of lines of code without alienating your colleagues

Fred Tingaud Murex @FredTingaudDev

Clang Based Refactoring

How to refactor millions of lines of code without alienating your colleagues

Fred Tingaud Murex @FredTingaudDev


HOOVER
Give her a Hoover and you give her the best
Christmas morning
(and forever after)
she'll be happier
with a Hoover
HOOVER
HOOVER
P. S. to husbands:
THE HOOVER COMPANY
North Canton, Ohio

Better Tooling

Continuous tooling improvement

$\Downarrow$

Drastic UX disruption

The Boy Scout Rule

Always leave the code better than you found it.

The Boy Scout Rule

Always leave the code better than you found it.

- Rename unclear functions and variables
- Remove raw loops
- Simplify the code

The Boy Scout Rule

- Powerful on frequently changing areas
- Part of the "Business As Usual"
- Easy to merge

The Boy Scout Rule

But, not convenient for:

- widely used APIs
- risky refactorings
- coordinated refactorings

Golden Boy Refactoring

- Get "unlimited" budget
- Stop everything and refactor

Golden Boy Refactoring

- Can tackle very complex refactorings
- Can lead to great improvements

Golden Boy Refactoring

But…

- Repetitive
- Error prone
- Whack-a-mole

Clang Tidy

Clang Tidy

C++ linter tool

Clang tool

Static analysis

Automatic fix

Clang

Objective C	⇘		⇗	x86_64
C++	⇒	AST* ⇒ ✨	⇒	arm
C	⇗		⇘	…

*Abstract Syntax Tree

```
int getAnswer()
{
    int compute = 2 * 21;
    return compute;
}
```

```
TranslationUnitDecl
`-FunctionDecl ⟨line:1:1, line:4:1⟩ line:1:5 getAnswer 'int ()'
  `-CompoundStmt ⟨col:17, line:4:1⟩
    |-DeclStmt ⟨line:2:5, col:25⟩
    | `-VarDecl ⟨col:5, col:23⟩ col:9 used compute 'int' cinit
    |   `-BinaryOperator ⟨col:19, col:23⟩ 'int' '*'
    |     |-IntegerLiteral ⟨col:19⟩ 'int' 2
    |     `-IntegerLiteral ⟨col:23⟩ 'int' 21
    `-ReturnStmt ⟨line:3:5, col:12⟩
      `-ImplicitCastExpr ⟨col:12⟩ 'int' ⟨LValueToRValue⟩
        `-DeclRefExpr ⟨col:12⟩ 'int' lvalue Var 0x55e0335b3f80 'compute' 'int'
```

```
int getAnswer()
{
    int compute = 2 * 21;
    return compute;
}
```

```
TranslationUnitDecl
`-FunctionDecl ⟨line:1:1, line:4:1⟩ line:1:5 getAnswer 'int ()'
  `-CompoundStmt ⟨col:17, line:4:1⟩
    |-DeclStmt ⟨line:2:5, col:25⟩
    | `-VarDecl ⟨col:5, col:23⟩ col:9 used compute 'int' cinit
    |   `-BinaryOperator ⟨col:19, col:23⟩ 'int' '*'
    |     |-IntegerLiteral ⟨col:19⟩ 'int' 2
    |     `-IntegerLiteral ⟨col:23⟩ 'int' 21
    `-ReturnStmt ⟨line:3:5, col:12⟩
      `-ImplicitCastExpr ⟨col:12⟩ 'int' ⟨LValueToRValue⟩
        `-DeclRefExpr ⟨col:12⟩ 'int' lvalue Var 0x55e0335b3f80 'compute' 'int'
```

```
int getAnswer()
{
    int compute = 2 * 21;
    return compute;
}
```

```
TranslationUnitDecl
`-FunctionDecl ⟨line:1:1, line:4:1⟩ line:1:5 getAnswer 'int ()'
  `-CompoundStmt ⟨col:17, line:4:1⟩
    |-DeclStmt ⟨line:2:5, col:25⟩
    | `-VarDecl ⟨col:5, col:23⟩ col:9 used compute 'int' cinit
    |   `-BinaryOperator ⟨col:19, col:23⟩ 'int' '*'
    |     |-IntegerLiteral ⟨col:19⟩ 'int' 2
    |     `-IntegerLiteral ⟨col:23⟩ 'int' 21
    `-ReturnStmt ⟨line:3:5, col:12⟩
      `-ImplicitCastExpr ⟨col:12⟩ 'int' ⟨LValueToRValue⟩
        `-DeclRefExpr ⟨col:12⟩ 'int' lvalue Var 0x55e0335b3f80 'compute' 'int'
```

```
int getAnswer()
{
    int compute = 2 * 21;
    return compute;
}
```

```
TranslationUnitDecl
`-FunctionDecl ⟨line:1:1, line:4:1⟩ line:1:5 getAnswer 'int ()'
  `-CompoundStmt ⟨col:17, line:4:1⟩
    |-DeclStmt ⟨line:2:5, col:25⟩
    | `-VarDecl ⟨col:5, col:23⟩ col:9 used compute 'int' cinit
    |   `-BinaryOperator ⟨col:19, col:23⟩ 'int' '*'
    |     |-IntegerLiteral ⟨col:19⟩ 'int' 2
    |     `-IntegerLiteral ⟨col:23⟩ 'int' 21
    `-ReturnStmt ⟨line:3:5, col:12⟩
      `-ImplicitCastExpr ⟨col:12⟩ 'int' ⟨LValueToRValue⟩
        `-DeclRefExpr ⟨col:12⟩ 'int' lvalue Var 0x55e0335b3f80 'compute' 'int'
```

```
int getAnswer()
{
    int compute = 2 * 21;
    return compute;
}
```

```
TranslationUnitDecl
`-FunctionDecl ⟨line:1:1, line:4:1⟩ line:1:5 getAnswer 'int ()'
  `-CompoundStmt ⟨col:17, line:4:1⟩
    |-DeclStmt ⟨line:2:5, col:25⟩
    | `-VarDecl ⟨col:5, col:23⟩ col:9 used compute 'int' cinit
    |   `-BinaryOperator ⟨col:19, col:23⟩ 'int' '*'
    |     |-IntegerLiteral ⟨col:19⟩ 'int' 2
    |     `-IntegerLiteral ⟨col:23⟩ 'int' 21
    `-ReturnStmt ⟨line:3:5, col:12⟩
      `-ImplicitCastExpr ⟨col:12⟩ 'int' ⟨LValueToRValue⟩
        `-DeclRefExpr ⟨col:12⟩ 'int' lvalue Var 0x55e0335b3f80 'compute' 'int'
```

```
int getAnswer()
{
    int compute = 2 * 21;
    return compute;
}
```

```
TranslationUnitDecl
`-FunctionDecl ⟨line:1:1, line:4:1⟩ line:1:5 getAnswer 'int ()'
  `-CompoundStmt ⟨col:17, line:4:1⟩
    |-DeclStmt ⟨line:2:5, col:25⟩
    | `-VarDecl ⟨col:5, col:23⟩ col:9 used compute 'int' cinit
    |   `-BinaryOperator ⟨col:19, col:23⟩ 'int' '*'
    |     |-IntegerLiteral ⟨col:19⟩ 'int' 2
    |     `-IntegerLiteral ⟨col:23⟩ 'int' 21
    `-ReturnStmt ⟨line:3:5, col:12⟩
      `-ImplicitCastExpr ⟨col:12⟩ 'int' ⟨LValueToRValue⟩
        `-DeclRefExpr ⟨col:12⟩ 'int' lvalue Var 0x55e0335b3f80 'compute' 'int'
```

```
int getAnswer()
{
    int compute = 2 * 21;
    return compute;
}
```

```
TranslationUnitDecl
`-FunctionDecl 〈line:1:1, line:4:1〉 line:1:5 getAnswer 'int ()'
  `-CompoundStmt 〈col:17, line:4:1〉
    |-DeclStmt 〈line:2:5, col:25〉
    | `-VarDecl 〈col:5, col:23〉 col:9 used compute 'int' cinit
    |   `-BinaryOperator 〈col:19, col:23〉 'int' '*'
    |     |-IntegerLiteral 〈col:19〉 'int' 2
    |     `-IntegerLiteral 〈col:23〉 'int' 21
    `-ReturnStmt 〈line:3:5, col:12〉
      `-ImplicitCastExpr 〈col:12〉 'int' 〈LValueToRValue〉
        `-DeclRefExpr 〈col:12〉 'int' lvalue Var 0x55e0335b3f80 'compute' 'int'
```

Clang

Objective C	⇘⇖		⇗	x86_64
C++	⇔	AST* ⇒ ✨	⇒	arm
C	⇗⇙		⇘	…

*Abstract Syntax Tree

Clang Tidy

```
for (vector<int>::iterator it = v.begin()
    ; it != v.end()
    ; ++it)
  cout << *it;
```

becomes

```
for (auto & elem : v)
  cout << elem;
```

```
void f(int x) {
  std::auto_ptr<int> a(new int(x));
  std::auto_ptr<int> b = a;
  take_ownership_fn(b);
}
```

becomes

```
void f(int x) {
  std::unique_ptr<int> a(new int(x));
  std::unique_ptr<int> b = std::move(a);
  take_ownership_fn(std::move(b));
}
```

Clang Tidy

C++ API for writing new checks

Clang Tidy Matcher

```
virtual void registerMatchers(MatchFinder* Finder)
{
    Finder->addMatcher(MY_MATCHER, this);
}
```

```
virtual void check(
    const MatchFinder::MatchResult& R) {
    // React to match
}
```

https://clang.llvm.org/docs/LibASTMatchersReference.html

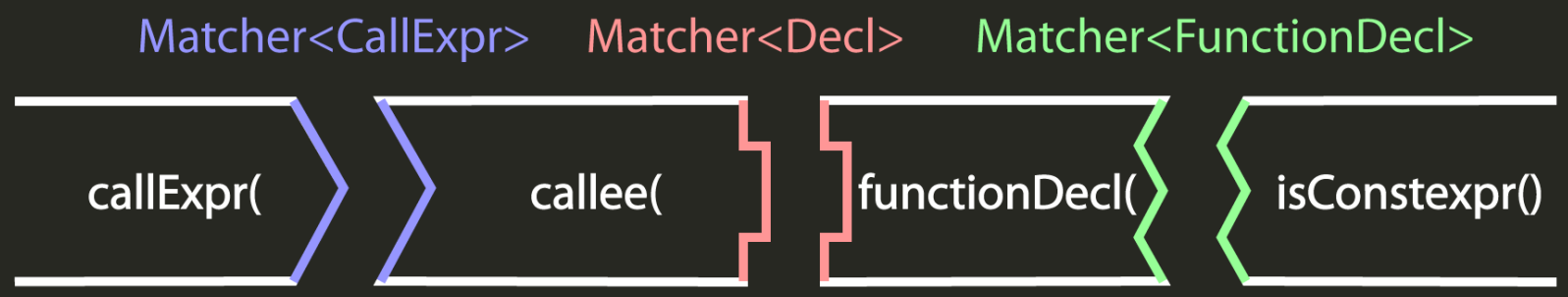
Matcher<CXXDependentScopeMemberExpr>	hasObjectExpression	Matcher<Expr> InnerMatcher
Matcher<CXXForRangeStmt>	hasBody	Matcher<Stmt> InnerMatcher

```
Matches a 'for', 'while', 'do while' statement or a function
definition that has a given body.

Given
  for (;;) {}
hasBody(compoundStmt())
  matches 'for (;;) {}'
with compoundStmt()
  matching '{}'
```

Matcher<CXXForRangeStmt>	hasLoopVariable	Matcher<VarDecl> InnerMatcher
Matcher<CXXForRangeStmt>	hasRangeInit	Matcher<Expr> InnerMatcher
Matcher<CXXMemberCallExpr>	onImplicitObjectArgument	Matcher<Expr> InnerMatcher
Matcher<CXXMemberCallExpr>	on	Matcher<Expr> InnerMatcher
Matcher<CXXMemberCallExpr>	thisPointerType	Matcher<Decl> InnerMatcher
Matcher<CXXMemberCallExpr>	thisPointerType	Matcher<QualType> InnerMatcher
Matcher<CXXMethodDecl>	forEachOverridden	Matcher<CXXMethodDecl> InnerMatcher
Matcher<CXXMethodDecl>	ofClass	Matcher<CXXRecordDecl> InnerMatcher
Matcher<CXXNewExpr>	hasArraySize	Matcher<Expr> InnerMatcher

Clang Tidy Matcher



Clang Tidy Custom Rule

```
if (a && b) {/* ... */}
```

vs

```
if (a and b) {/* ... */}
```

Clang Tidy Matcher

```
virtual void registerMatchers(MatchFinder* Finder)
{
    Finder->addMatcher(
        binaryOperator(hasOperatorName("&&"))
            .bind("my_op")
        , this);
}
```

Clang Tidy Matcher

```
virtual void registerMatchers(MatchFinder* Finder)
{
    Finder->addMatcher(
        binaryOperator(hasOperatorName("&&"))
            .bind("my_op")
        , this);
}
```

Clang Tidy Matcher

```
virtual void registerMatchers(MatchFinder* Finder)
{
    Finder->addMatcher(
        binaryOperator(hasOperatorName("&&"))
            .bind("my_op")
        , this);
}
```

Clang Tidy Matcher

```
virtual void registerMatchers(MatchFinder* Finder)
{
    Finder->addMatcher(
        binaryOperator(hasOperatorName("&&"))
            .bind("my_op")
        , this);
}
```

Clang Tidy Matcher

```
virtual void registerMatchers(MatchFinder* Finder)
{
    Finder->addMatcher(
        binaryOperator(hasOperatorName("&&"))
            .bind("my_op")
        , this);
}
```

Clang Tidy Diagnostic

```
virtual void check(
    const MatchFinder::MatchResult& R)
{
    auto Op = R.Nodes.getNodeAs<BinaryOperator>(
        "my_op");
    auto Diag = diag(op->getOperatorLoc(),
        "'&&' can be replaced by 'and'");
    Diag << FixItHint::CreateReplacement(
        op->getOperatorLoc(), "and");
}
```

Clang Tidy Diagnostic

```
virtual void check(
    const MatchFinder::MatchResult& R)
{
    auto Op = R.Nodes.getNodeAs<BinaryOperator>(
        "my_op");
    auto Diag = diag(op->getOperatorLoc(),
        "'&&' can be replaced by 'and'");
    Diag << FixItHint::CreateReplacement(
        op->getOperatorLoc(), "and");
}
```

Clang Tidy Diagnostic

```
virtual void check(
    const MatchFinder::MatchResult& R)
{
    auto Op = R.Nodes.getNodeAs<BinaryOperator>(
        "my_op");
    auto Diag = diag(op->getOperatorLoc(),
        "'&&' can be replaced by 'and'");
    Diag << FixItHint::CreateReplacement(
        op->getOperatorLoc(), "and");
}
```

Clang Tidy Diagnostic

```
virtual void check(
    const MatchFinder::MatchResult& R)
{
    auto Op = R.Nodes.getNodeAs<BinaryOperator>(
        "my_op");
    auto Diag = diag(op->getOperatorLoc(),
        "'&&' can be replaced by 'and'");
    Diag << FixItHint::CreateReplacement(
        op->getOperatorLoc(), "and");
}
```

Clang Tidy Output

```
$ clang-tidy -checks=-*,misc-cppcon demo.cpp
1 warning generated.
I:\demo.cpp:2:12: warning: '&&' can be replaced by 'and' [misc-cppcon]
  return a && b;
           ^~
           and
```

Unused parameters

```
void displayScreen(std::string const& name,
                   int param1,
                   int unused,
                   bool param3);
```

Unused parameters

```
virtual void registerMatchers(MatchFinder* Finder){
 Finder->addMatcher(callExpr(callee(
  functionDecl(hasName("displayScreen"),
               parameterCountIs(4))))
  .bind("function_to_clean"),
  this);
}
```

Unused parameters

```
virtual void registerMatchers(MatchFinder* Finder){
 Finder->addMatcher(callExpr(callee(
  functionDecl(hasName("displayScreen"),
               parameterCountIs(4))))
  .bind("function_to_clean"),
  this);
}
```

Unused parameters

```
virtual void registerMatchers(MatchFinder* Finder){
 Finder->addMatcher(callExpr(callee(
  functionDecl(hasName("displayScreen"),
               parameterCountIs(4))))
  .bind("function_to_clean"),
  this);
}
```

Unused parameters

```
virtual void registerMatchers(MatchFinder* Finder){
 Finder->addMatcher(callExpr(callee(
 functionDecl(hasName("displayScreen"),
                parameterCountIs(4))))
  .bind("function_to_clean"),
  this);
}
```

Unused parameters

```
virtual void registerMatchers(MatchFinder* Finder){
 Finder->addMatcher(callExpr(callee(
  functionDecl(hasName("displayScreen"),
               parameterCountIs(4)))
  .bind("function_to_clean"),
  this);
}
```

Unused parameters

```
virtual void registerMatchers(MatchFinder* Finder){
 Finder->addMatcher(callExpr(callee(
  functionDecl(hasName("displayScreen"),
               parameterCountIs(4))))
  .bind("function_to_clean"),
  this);
}
```

Unused parameters

```
virtual void check(
    const MatchFinder::MatchResult& R)
{
  auto Call = R.Nodes.getNodeAs<CallExpr>(
      "function_to_clean");
  auto Param = call->getArg(2);
  auto Diag = diag(param->getLocation(),
      "This parameter is unused");
  Diag << FixItHint::CreateRemoval(
      param->getLocation());
}
```

Unused parameters

```
virtual void check(
    const MatchFinder::MatchResult& R)
{
  auto Call = R.Nodes.getNodeAs<CallExpr>(
      "function_to_clean");
  auto Param = call->getArg(2);
  auto Diag = diag(param->getLocation(),
      "This parameter is unused");
  Diag << FixItHint::CreateRemoval(
      param->getLocation());
}
```

Unused parameters

```
virtual void check(
    const MatchFinder::MatchResult& R)
{
  auto Call = R.Nodes.getNodeAs<CallExpr>(
      "function_to_clean");
  auto Param = call->getArg(2);
  auto Diag = diag(param->getLocation(),
      "This parameter is unused");
  Diag << FixItHint::CreateRemoval(
      param->getLocation());
}
```

Unused parameters

```
virtual void check(
    const MatchFinder::MatchResult& R)
{
  auto Call = R.Nodes.getNodeAs<CallExpr>(
      "function_to_clean");
  auto Param = call->getArg(2);
  auto Diag = diag(param->getLocation(),
      "This parameter is unused");
  Diag << FixItHint::CreateRemoval(
      param->getLocation());
}
```

Unused parameters

```
virtual void check(
    const MatchFinder::MatchResult& R)
{
  auto Call = R.Nodes.getNodeAs<CallExpr>(
      "function_to_clean");
  auto Param = call->getArg(2);
  auto Diag = diag(param->getLocation(),
      "This parameter is unused");
  Diag << FixItHint::CreateRemoval(
      param->getLocation());
}
```

Magic Numbers

```
class Reader {
  /// 0 on success, 1 on failure
  virtual int read(Message const& m) = 0;
};
```

```
int SomeReader::read(Message const& m)
{
  int result = 0;
  if (!readHeader(m))
    return 1;
  if (m.id > 0)
    result = m.id < 10
        ? readBody(m)
        : 1;
  return result;
}
```

```
int SomeReader::read(Message const& m)
{
  int result = 0;
  if (!readHeader(m))
    return 1;
  if (m.id > 0)
    result = m.id < 10
        ? readBody(m)
        : 1;
  return result;
}
```

```
int SomeReader::read(Message const& m)
{
  int result = 0;
  if (!readHeader(m))
    return 1;
  if (m.id > 0)
    result = m.id < 10
        ? readBody(m)
        : 1;
  return result;
}
```

```
int SomeReader::read(Message const& m)
{
  int result = 0;
  if (!readHeader(m))
    return 1;
  if (m.id > 0)
    result = m.id < 10
         ? readBody(m)
         : 1;
  return result;
}
```

```
Result SomeReader::read(Message const& m)
{
  Result result = Result::Success;
  if (!readHeader(m))
    return Result::Failure;
  if (m.id > 0)
    result = m.id < 10
        ? readBody(m)
        : Result::Failure;
  return result;
}
```

Merging / Integrating

- Can rerun the script on conflicts
- Can revert and come back later on problem

New Possibilities

- Unsolvable problems became solvable
- We can refactor *at large scale*
- We can *revolutionize* our code!

Thank you

** Record scratch **

Do you want a revolution?

Huge changelists

- Impossible to validate
- Impossible to review by code owners

C++ is HARD!

C++ is HARD!

C++ is HARD!

So is AST matching!

C++ is HARD!

So is AST matching!

You will trip more than once.

AST matching is hard

```
virtual void registerMatchers(MatchFinder* Finder)
{
    Finder->addMatcher(
        binaryOperator(hasOperatorName("&&"))
            .bind("my_op")
        , this);
}
```

AST matching is hard

```
virtual void registerMatchers(MatchFinder* Finder)
{
    Finder->addMatcher(
        binaryOperator(hasOperatorName("&&"))
            .bind("my_op")
        , this);
}
```

`&&` and `and` are both caught

AST matching is hard

```
virtual void registerMatchers(MatchFinder* Finder)
{
    Finder->addMatcher(
        binaryOperator(hasOperatorName("&&"))
            .bind("my_op")
        , this);
}
```

`&&` and `and` are both caught

`bool operator&&(T const&)` is not caught

AST matching is hard

```
void displayScreen(std::string const& name,
                   int param1,
                   int unused,
                   bool param3);
```

AST matching is hard

```
#define FALSE 0

displayScreen("Hello", t1, FALSE, t3);
```

AST matching is hard

```
#define FALSE 0

displayScreen("Hello", t1, FALSE, t3);
```

AST matching is hard

```
#define FALSE

displayScreen("Hello", t1, FALSE, t3);
```

AST matching is hard

Enum class conversion

AST matching is hard

Enum class conversion

Lambdas

AST matching is hard

Enum class conversion

Lambdas

Templated classes

AST matching is hard

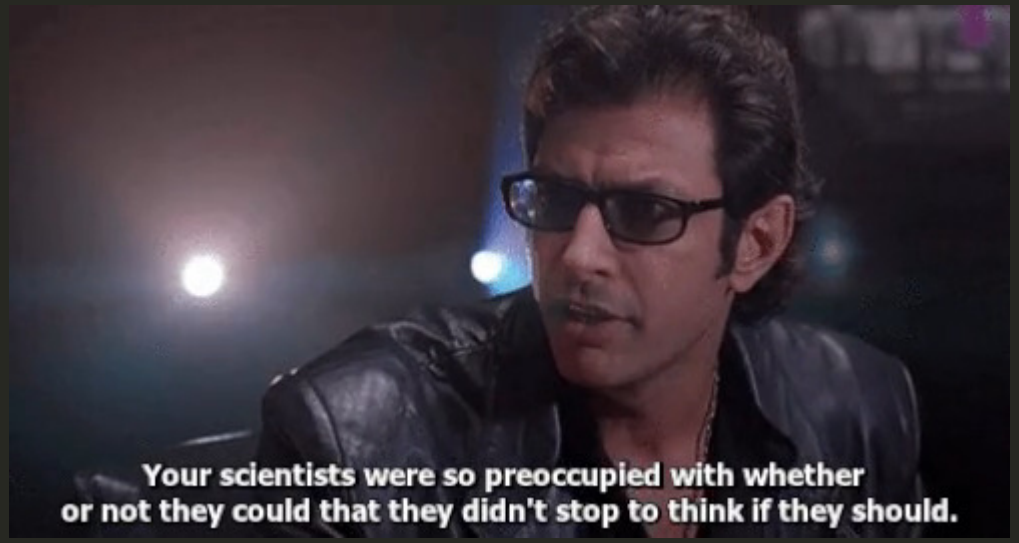
Not really a problem…

AST matching is hard

Not really a problem…

If you can see it!

Readable pull requests



An incremental approach

```
void displayScreen(std::string const& name,
                   int param1,
                   bool param3);

void displayScreen(std::string const& name,
                   int param1,
                   int unused,
                   bool param3)
{ displayScreen(name, param1, param3); }
```

An incremental approach

```
class Reader {
  /// 0 on success, 1 on failure
  virtual int read(Message const& m) = 0;
};
```

An incremental approach

```
class Reader {
  /// 0 on success, 1 on failure
  virtual int read(Message const& m) = 0;

  enum class Result {Success, Failure};
  virtual Result readEnum(Message const& m) {}
};
```

An incremental approach

```
class SomeReader {
  int read(Message const& m) override {
    // Old implementation
  }
};
```

An incremental approach

```
class SomeReader {
  Result readEnum(Message const& m) {
    // New implementation
  }

  int read(Message const& m) override {
    return readEnum(m) == Result::Success ? 0 : 1;
  }
};
```

An incremental approach

```
class Reader {
  /// 0 on success, 1 on failure
  virtual int read(Message const& m) = 0;

  enum class Result {Success, Failure};
  virtual Result readEnum(Message const& m) {}
};
```

An incremental approach

```
class Reader {
  /// 0 on success, 1 on failure
  virtual int read(Message const& m) = 0;

  enum class Result {Success, Failure};
  virtual Result readEnum(Message const& m) {}
};
```

An incremental approach

```
class Reader {
  enum class Result {Success, Failure};
  virtual Result readEnum(Message const& m) = 0;
};
```

An incremental approach

```
class Reader {
  enum class Result {Success, Failure};
  virtual Result readEnum(Message const& m) = 0;

  virtual Result read(Message const& m) {}
};
```

An incremental approach

More work…

Find the right pull request size

1 PR of 1000 files?

1000 PR of 1 file?

Automatize

MANY

- patches
- pull-requests
- mails
- merges

Protect

What if other teams are undoing your work?

Protect

What if other teams are undoing your work?

[[deprecated]]

Protect

What if other teams are undoing your work?

[[deprecated]]

Clang-tidy!

Finish

Temporary increase of complexity

Clang Tidy

- Redefines our job
- Requires a sustainable approach
- Highlights higher level problems
- Opens new possibilities

Clang Based Refactoring

How to refactor millions of lines of code without alienating your colleagues

Fred Tingaud Murex @FredTingaudDev