

# The Untapped Potential of Software Visualization

by Eberhard Gräther  
C++Now 2018



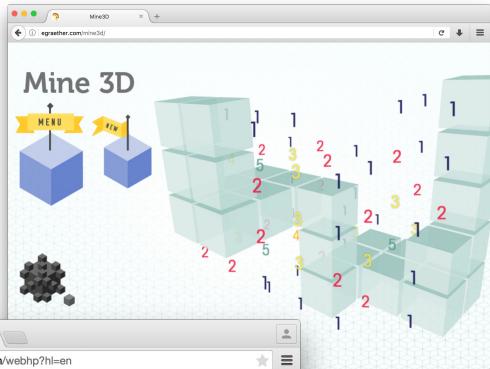
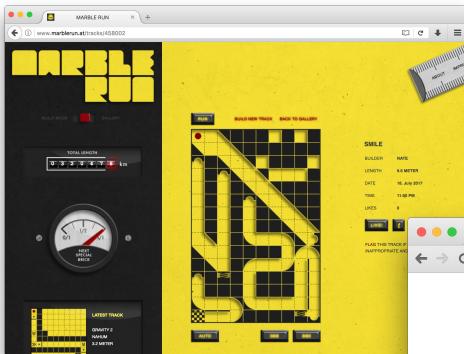
# Introduction

---

0/5



# Work



The collage includes:

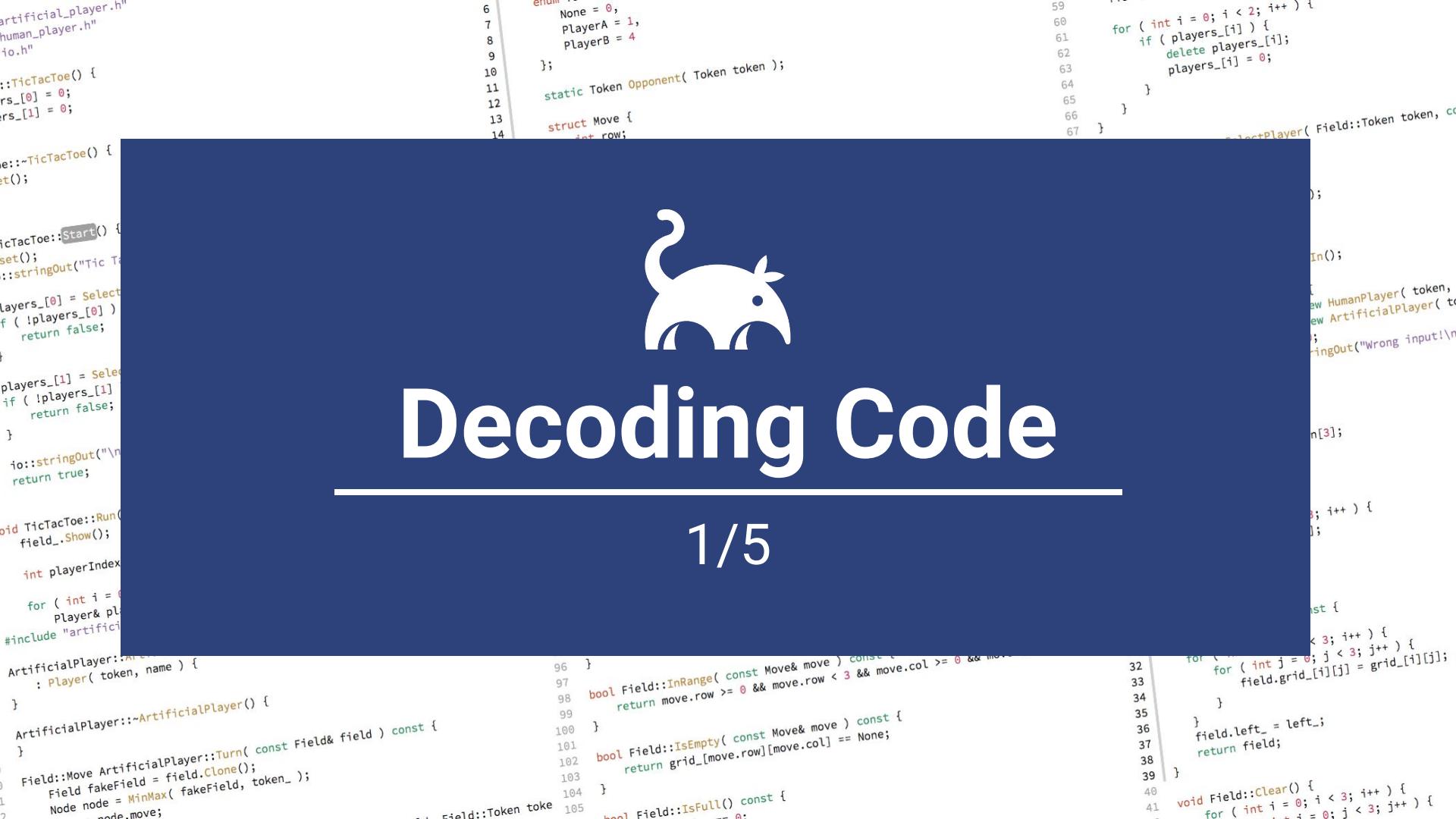
- A screenshot of a game gallery titled "Playin" showing various mini-games like "mine festiv", "secret win", and "which rocket will make you lose?".
- An IDE interface for "TicTacToe:Run" with code in C++ and a UML class diagram.
- A screenshot of the "chrome://tracing" page in Chrome DevTools showing a timeline of rendering tasks.
- A screenshot of a browser window showing a Google search result for "Google" with GPU information: FPS: 55.7, GPU raster: off (device), GPU memory: 6.2 MB used, 512.0 MB max.
- A screenshot of a browser window showing a game titled "Mine 3D" with a 3D cube puzzle.
- A screenshot of a browser window showing a game titled "Marble Run" with a marble track.
- A screenshot of a browser window showing a game titled "esprimo - soft body painter" with a 3D model being painted.
- A screenshot of a browser window showing a game titled "esprimo - soft body painter" with a 3D model being painted.



*“Programmers tend to adapt to the level of representation **provided by the computer**, instead of adapting the computers representations to their **perceptive abilities**.”*

- Stephan Dhiel

*(Software Visualization - Visualizing the Structure, Behaviour, and Evolution of Software)*



# Decoding Code

1/5

```
artificial_player.h
human_player.h
io.h"
};

::TicTacToe() {
    rs_[0] = 0;
    rs_[1] = 0;
}

::~TicTacToe() {
    ~Tic();
}

TicTacToe::Start() {
    set();
    ::stringOut("Tic Ta
layers_[0] = Select
f ( !players_[0] )
    return false;
}

players_[1] = Select
if ( !players_[1] )
    return false;
}

io::stringOut("\n")
return true;
}

void TicTacToe::Run()
{
    field_.Show();
    int playerIndex
    for ( int i = 0; i < 2; i++ ) {
        Player& pl
        #include "artificial_player.h"
        ArtificialPlayer::ArtificialPlayer( const Player& token, name )
        : Player( token, name ) {
    }

    ArtificialPlayer::~ArtificialPlayer()
    }

    Field::Move ArtificialPlayer::Turn( const Field& field ) const {
        Field fakefield = field.Clone();
        Node node = MinMax( fakefield, token_ );
        node.move;
        Field::Token take
    }
}

enum Token {
    None = 0,
    PlayerA = 1,
    PlayerB = 4
};

static Token Opponent( Token token );
struct Move {
    int row;
};

for ( int i = 0; i < 2; i++ ) {
    if ( players_[i] ) {
        delete players_[i];
        players_[i] = 0;
    }
}

selectPlayer( Field::Token token, co
);

In();

HumanPlayer( token,
new ArtificialPlayer( token );
stringOut("Wrong input!\n"
n[3];

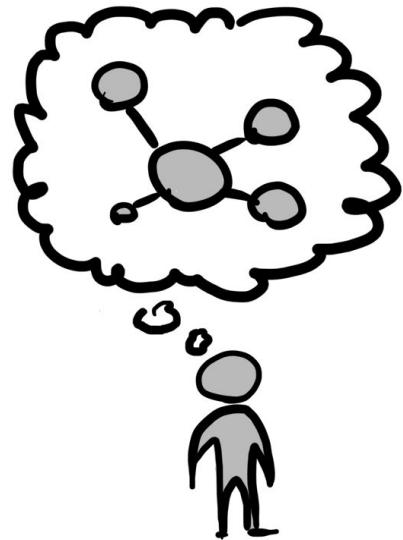
; i++ ) {
};

Field::InRange( const Move& move ) const {
    return move.row >= 0 && move.col < 3 && move.col >= 0 && move
};

bool Field::IsEmpty( const Move& move ) const {
    return grid_[move.row][move.col] == None;
};

bool Field::IsFull() const {
    for ( int i = 0; i < 3; i++ ) {
        for ( int j = 0; j < 3; j++ ) {
            if ( grid_[i][j] == None ) {
                field.left_ = left_;
                return field;
            }
        }
    }
}

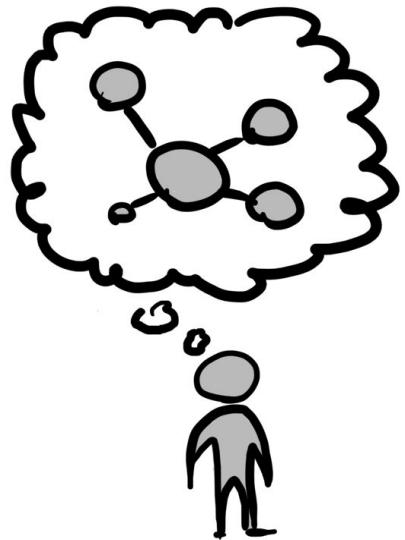
void Field::Clear() {
    for ( int i = 0; i < 3; i++ ) {
        for ( int j = 0; j < 3; j++ ) {
            grid_[i][j] = None;
        }
    }
}
```



&gt;



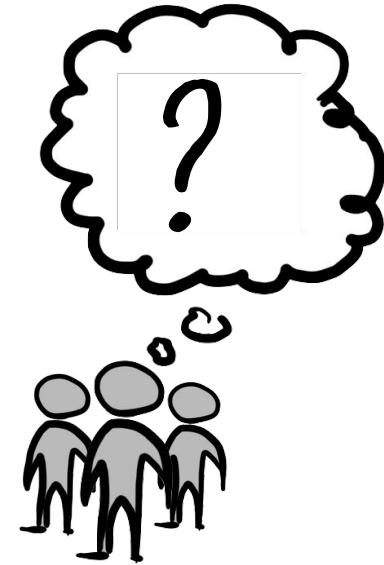
**WRITING**



&gt;



&gt;



WRITING

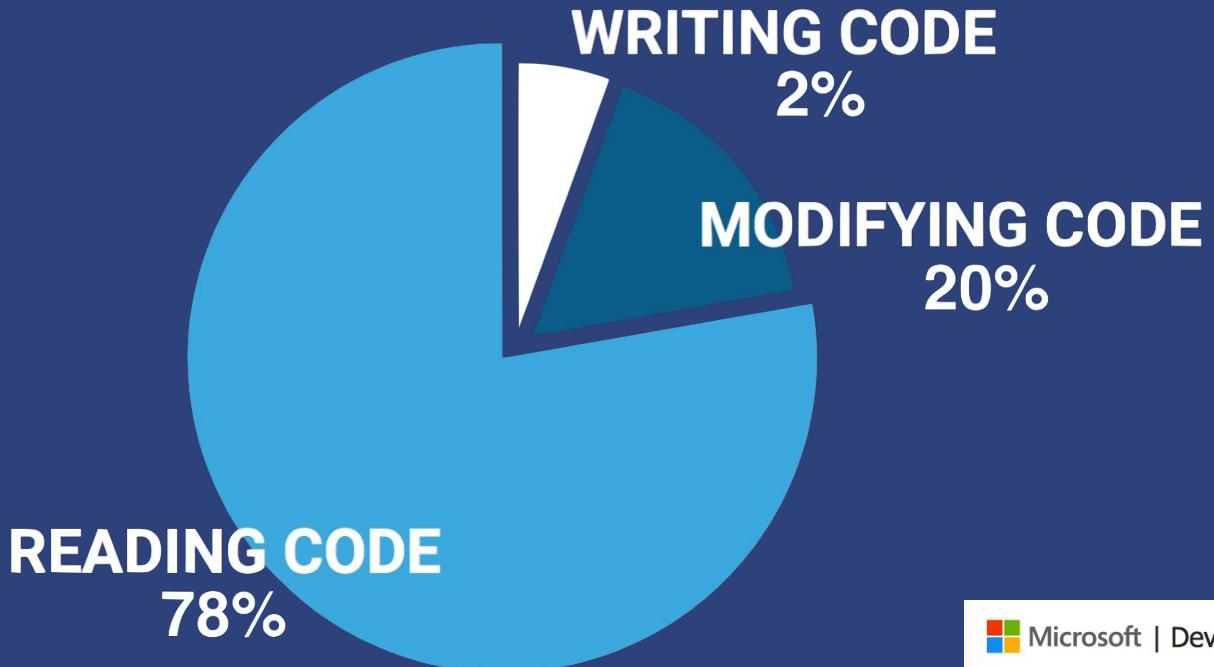
READING



***“Indeed, the ratio of time spent reading vs. writing is well over 10:1. We are constantly reading old code as part of the effort to write new code.”***

- Robert C. Martin

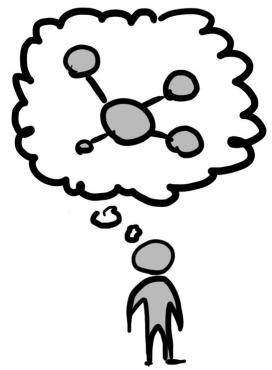
*(Clean Code: A Handbook of Agile Software Craftsmanship)*



Microsoft | Developer

Peter Hallam's WebLog

\* <https://blogs.msdn.microsoft.com/peterhal/2006/01/04/what-do-programmers-really-do-anyway-aka-part-2-of-the-yardstick-saga/>



&gt;



&gt;



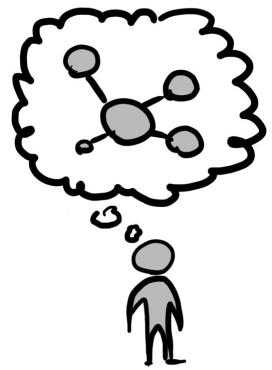
WRITING

READING



## Usual Advice

- Write tests
- Write documentation
- Follow coding guidelines
- Make code reviews
- Use standard C++
- Remove dead code
- Name symbols properly
- Spend time refactoring
- Separate responsibilities
- Use static analysis checks
- Fix compiler warnings
- Have well defined components
- Use runtime checks
- No magic numbers
- Follow a clear style convention
- Don't repeat yourself
- Avoid circular dependencies



&gt;

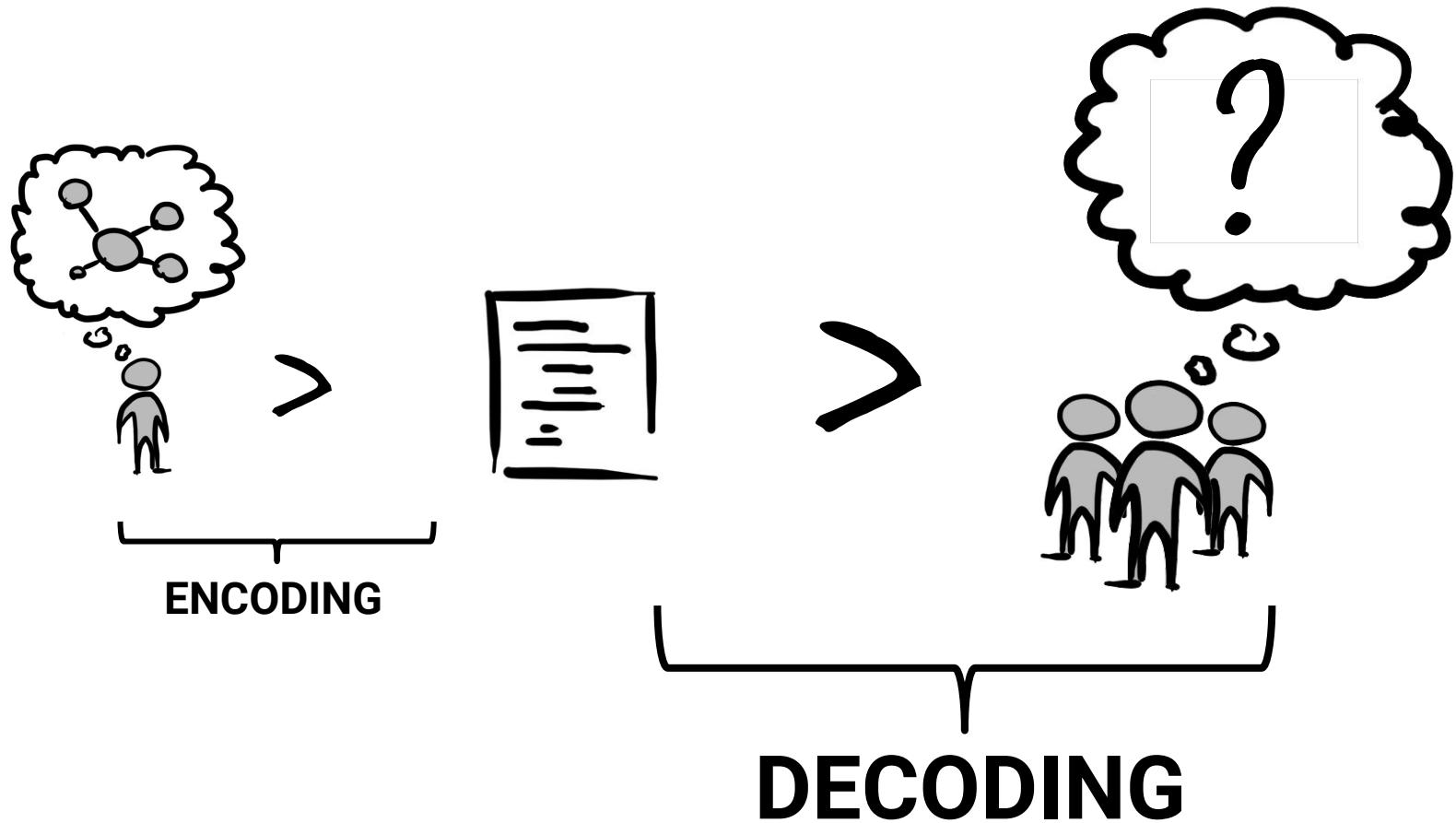


&gt;



WRITING

READING





# C++ source code is textual data with high information density.

```
1  template <typename T>
2  class Countable
3  {
4  public:
5      Countable() {
6          ++objectCount;
7      }
8
9
10     int getCount() const {
11         return objectCount;
12     }
13
14 private:
15     static int objectCount;
16 };
17
18 template <typename T> int Countable<T>::objectCount(0);
19
20 class Object : public Countable<Object>
21 {};
22
23 int main()
24 {
25     Object o;
26     int c = o.getCount();
27     return 0;
28 }
29
```



# C++ source code is textual data with high information density.

## For readability:

- syntax highlighting
- naming convention
- whitespace

```
1  template <typename T>
2  class Countable
3  {
4  public:
5      Countable() {
6          ++objectCount;
7      }
8
9
10     int getCount() const {
11         return objectCount;
12     }
13
14 private:
15     static int objectCount;
16 };
17
18 template <typename T> int Countable<T>::objectCount(0);
19
20 class Object : public Countable<Object>
21 {};
22
23 int main()
24 {
25     Object o;
26     int c = o.getCount();
27     return 0;
28 }
29
```



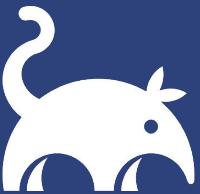
# C++ source code is textual data with high information density.

## For readability:

- syntax highlighting
- naming convention
- whitespace

## But not ideal for our mind!

```
1  template <typename T>
2  class Countable
3  {
4  public:
5      Countable() {
6          ++objectCount;
7      }
8
9
10     int getCount() const {
11         return objectCount;
12     }
13
14 private:
15     static int objectCount;
16 };
17
18 template <typename T> int Countable<T>::objectCount();
19
20 class Object : public Countable<Object>
21 {};
22
23 int main()
24 {
25     Object o;
26     int c = o.getCount();
27     return 0;
28 }
29
```



# Data Visualization

---

2/5



# Visual Variables

position:



size:



shape:



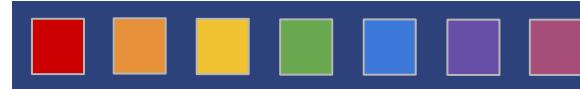
orientation:



value:



color:



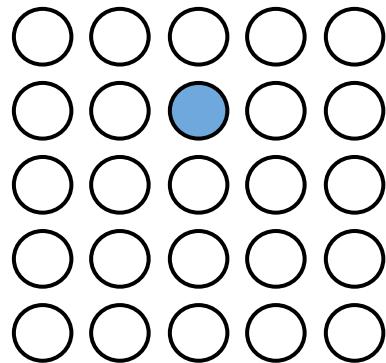
texture:



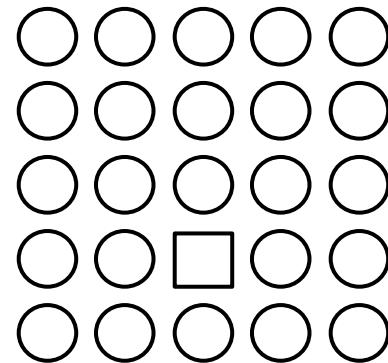


# Preattentive Perception

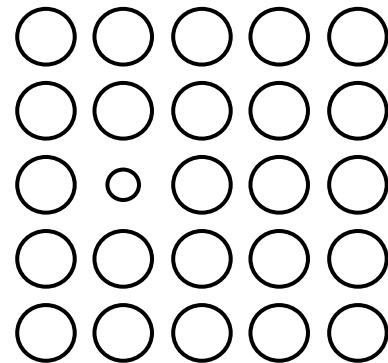
Features are perceived within 200 ms:



**Color**



**Shape**

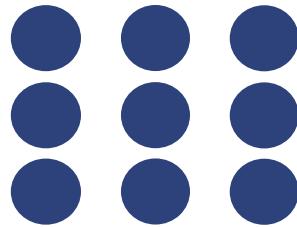


**Size**

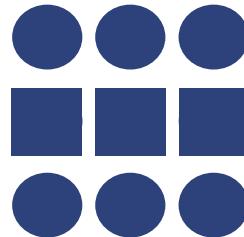
(*Software Visualization, Dhiel 2007, S. 18*)



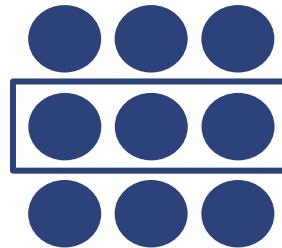
# Principles of Grouping



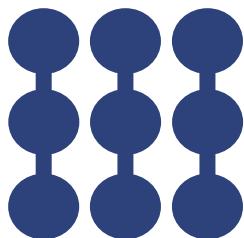
Proximity



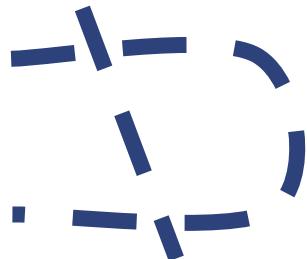
Similarity



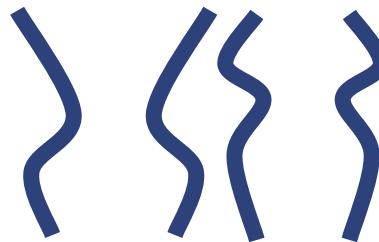
Enclosure



Connection



Continuation

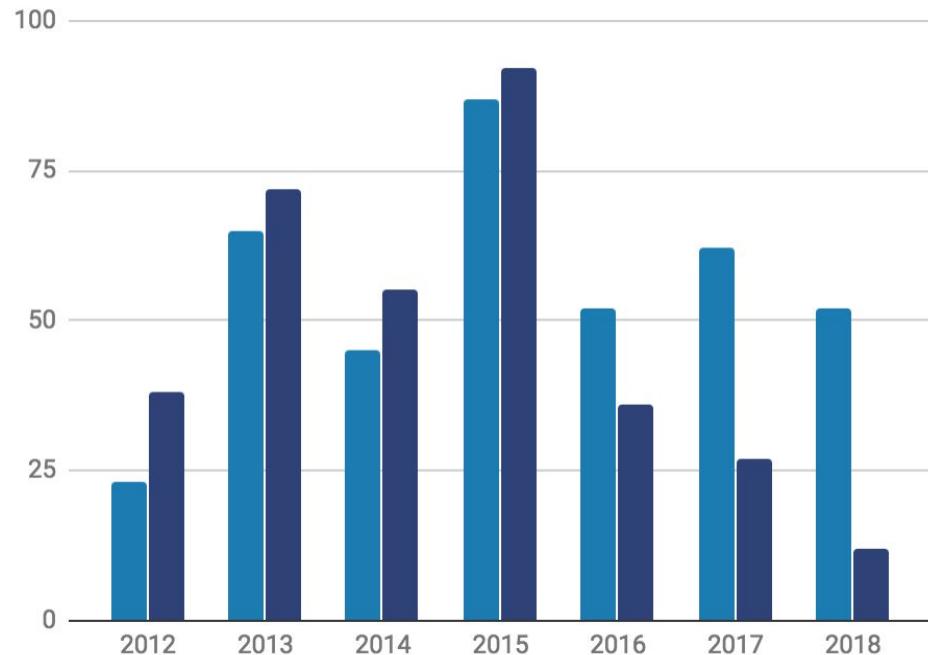


Symmetry



# Sample 1

2012	23	38
2013	65	72
2014	45	55
2015	87	92
2016	52	36
2017	62	27
2018	52	12





# Sample 2

```
1 template<typename T>class Countable{  
2     public:Countable(){++objectCount;}int  
3         getCount()const{return objectCount;}  
4         private:static int objectCount;;  
5         template<typename T>int Countable<T>::  
6             objectCount(0);class Object:public  
7             Countable<Object>{};int main(){Object  
8                 o;int c=o.getCount();return 0;}  
9  
10
```

```
1 template <typename T>  
2 class Countable  
3 {  
4     public:  
5         Countable() {  
6             ++objectCount;  
7         }  
8  
9  
10        int getCount() const {  
11            return objectCount;  
12        }  
13  
14        private:  
15            static int objectCount;  
16        };  
17  
18        template <typename T> int Countable<T>::objectCount(0);  
19  
20        class Object : public Countable<Object>  
21    {};  
22  
23        int main()  
24    {  
25        Object o;  
26        int c = o.getCount();  
27        return 0;  
28    }  
29
```



# Software Visualization

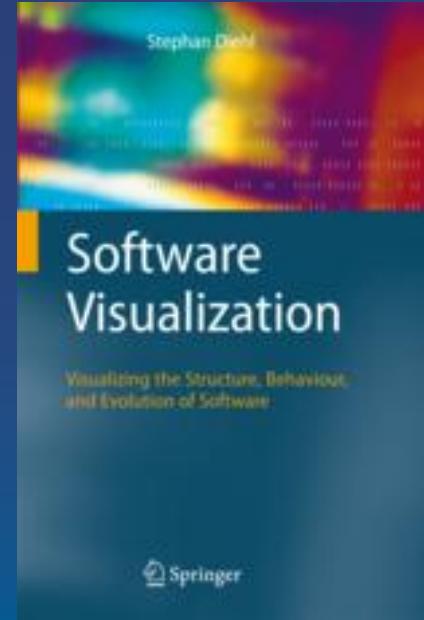
---

3/5



# Software Visualization

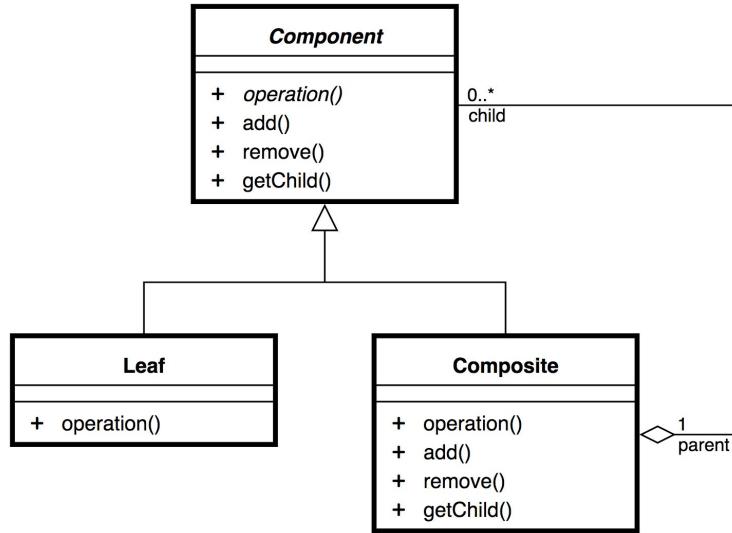
- **Structure:**  
Static Analysis - code is data
- **Behaviour:**  
Dynamic Analysis - data from execution
- **Evolution:**  
Version Control - history is data



*Software Visualization, Stephan Dhael, 2007*  
<https://www.springer.com/us/book/9783540465041>

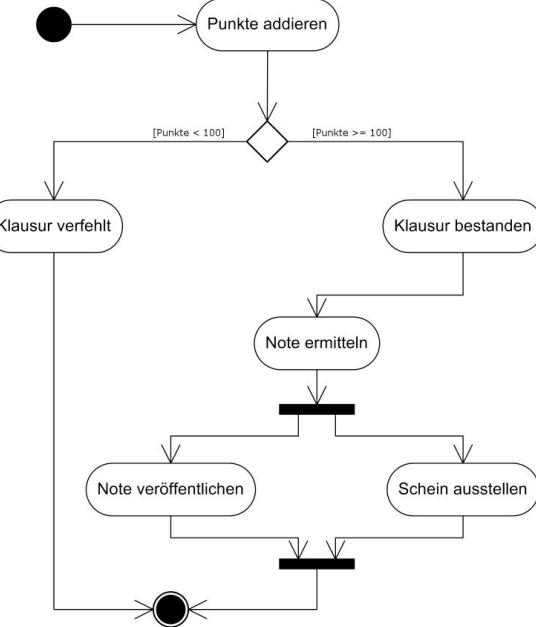


# UML



## Class Diagram

By Composite\_UML\_class\_diagram.svg: Trashtoyderivative work: « Aaron Rotenberg » Talk « (Composite\_UML\_class\_diagram.svg) [Public domain], via Wikimedia Commons  
([https://commons.wikimedia.org/wiki/File:Composite\\_UML\\_class\\_diagram\\_\(fixed\).svg](https://commons.wikimedia.org/wiki/File:Composite_UML_class_diagram_(fixed).svg))



## Activity Diagram

By Original PNG Activity diagram 2.png by Stern (Redrawn in SVG) [CC BY-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/>)], via Wikimedia Commons  
(<https://commons.wikimedia.org/wiki/File:Uml-Activity-Beispiel1.svg>)



# Structure Examples

Doxygen Inheritance and Collaboration Graphs:

[https://clang.llvm.org/doxygen/classclang\\_1\\_1NamedDecl.html](https://clang.llvm.org/doxygen/classclang_1_1NamedDecl.html)

Visual Studio Code Maps:

<https://docs.microsoft.com/en-us/visualstudio/modeling/map-dependencies-across-your-solutions>

<https://docs.microsoft.com/en-us/visualstudio/modeling/use-code-maps-to-debug-your-applications>

CppDepend Dependency Structure Matrix: [https://www.cppdepend.com/Doc\\_VS\\_Arch](https://www.cppdepend.com/Doc_VS_Arch)

CppDepend Code Metrics Treemap: [https://www.cppdepend.com/Doc\\_Treemap](https://www.cppdepend.com/Doc_Treemap)



# Behaviour Examples

VisuAlgo Sorting Algorithms: <https://visualgo.net/en/sorting>

Chrome Tracing Flame Chart: chrome://tracing/



# Evolution Examples

Git branch graph: `git log --oneline --graph`

CppDepend Trend Monitoring: [https://www.cppdepend.com/Doc\\_Trend](https://www.cppdepend.com/Doc_Trend)



# Visualizing C++

---

4/5



# Goal:

## Overview of existing symbols

## Dependencies between symbols

```
1  template <typename T>
2  class Countable
3  {
4  public:
5      Countable() {
6          ++objectCount;
7      }
8
9
10     int getCount() const {
11         return objectCount;
12     }
13
14 private:
15     static int objectCount;
16 };
17
18 template <typename T> int Countable<T>::objectCount(0);
19
20 class Object : public Countable<Object>
21 {};
22
23 int main()
24 {
25     Object o;
26     int c = o.getCount();
27     return 0;
28 }
29
```



## Symbols (nodes):

- Type
- Function (Method)
- Variable (Field)

## Relationships (edges):

- inherits
- specializes
- calls
- accesses

```
1  template <typename T>
2  class Countable
3  {
4  public:
5      Countable() {
6          ++objectCount;
7      }
8
9
10     int getCount() const {
11         return objectCount;
12     }
13
14 private:
15     static int objectCount;
16 };
17
18 template <typename T> int Countable<T>::objectCount();
19
20 class Object : public Countable<Object>
21 {};
22
23 int main()
24 {
25     Object o;
26     int c = o.getCount();
27     return 0;
28 }
29
```



# Nodes

Type

```
class Type { };
```

function

```
void function() { }
```

variable

```
int variable;
```

Class



PUBLIC

method



PRIVATE

field

```
class Class
{
public:
    void method();
private:
    int field;
};
```



# Edges



```
void function() { variable = 1; }
```



```
void functionA() { functionB(); }
```



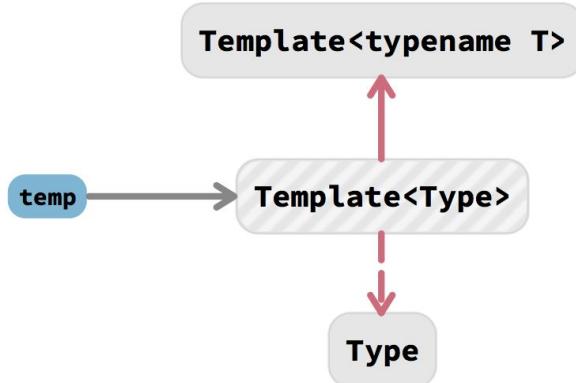
```
void function() { Type t; }
```



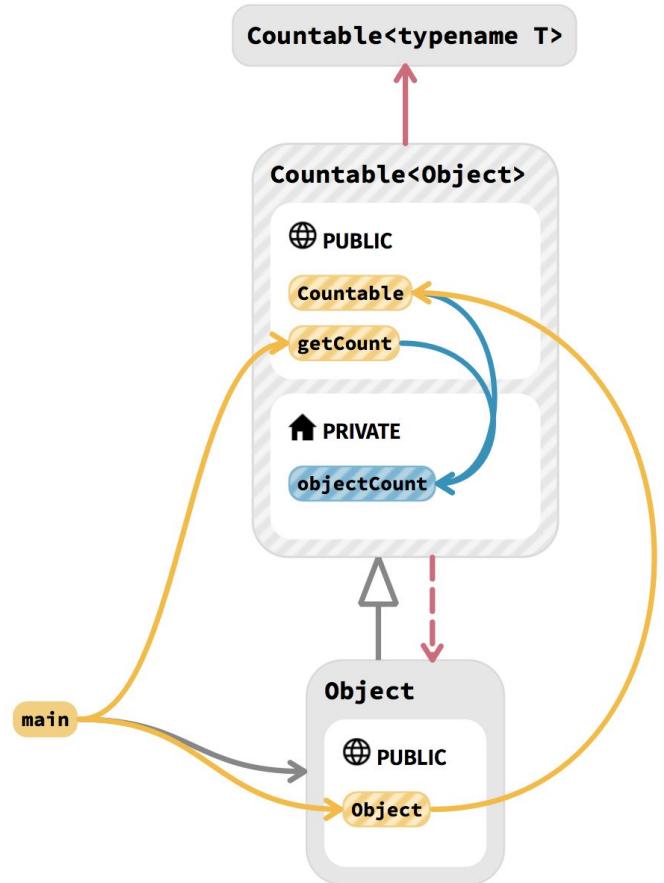
# Edges



```
class ClassA  
    : public ClassB  
{};
```



```
template <typename T>  
class Template  
{};  
Template<Type> temp;
```



```

1  template <typename T>
2  class Countable
3  {
4  public:
5      Countable() {
6          ++objectCount;
7      }
8
9
10     int getCount() const {
11         return objectCount;
12     }
13
14 private:
15     static int objectCount;
16 };
17
18 template <typename T> int Countable<T>::objectCount(0);
19
20 class Object : public Countable<Object>
21 {};
22
23 int main()
24 {
25     Object o;
26     int c = o.getCount();
27     return 0;
28 }
29
  
```



## Pros

- + Ground truth
- + Shows all information
- + Familiar syntax from writing

```
1  template <typename T>
2  class Countable
3  {
4  public:
5      Countable() {
6          ++objectCount;
7      }
8
9
10     int getCount() const {
11         return objectCount;
12     }
13
14 private:
15     static int objectCount;
16 };
17
18 template <typename T> int Countable<T>::objectCount(0);
19
20 class Object : public Countable<Object>
21 {};
22
23 int main()
24 {
25     Object o;
26     int c = o.getCount();
27     return 0;
28 }
29
```



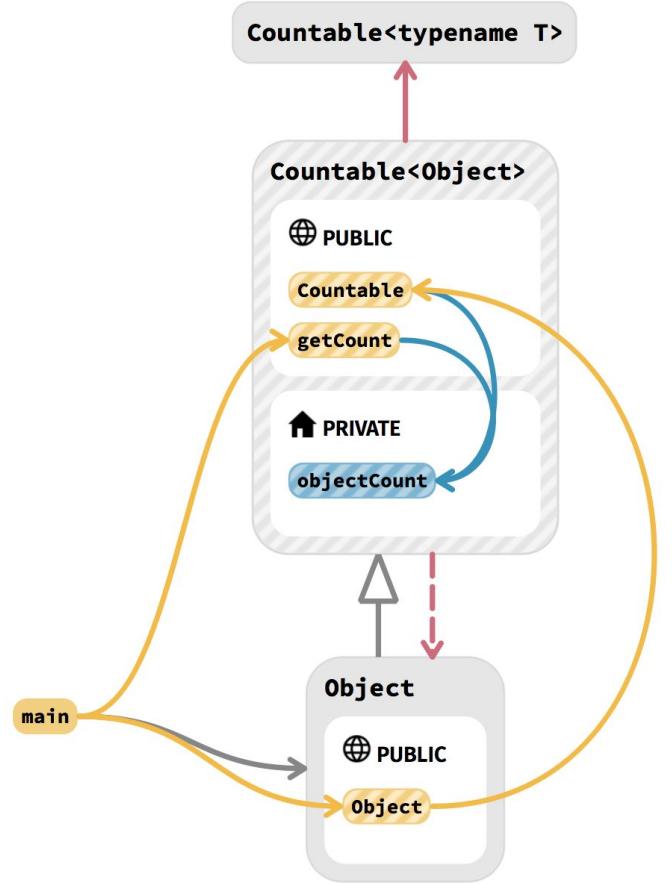
## Pros

- + Ground truth
- + Shows all information
- + Familiar syntax from writing

## Cons

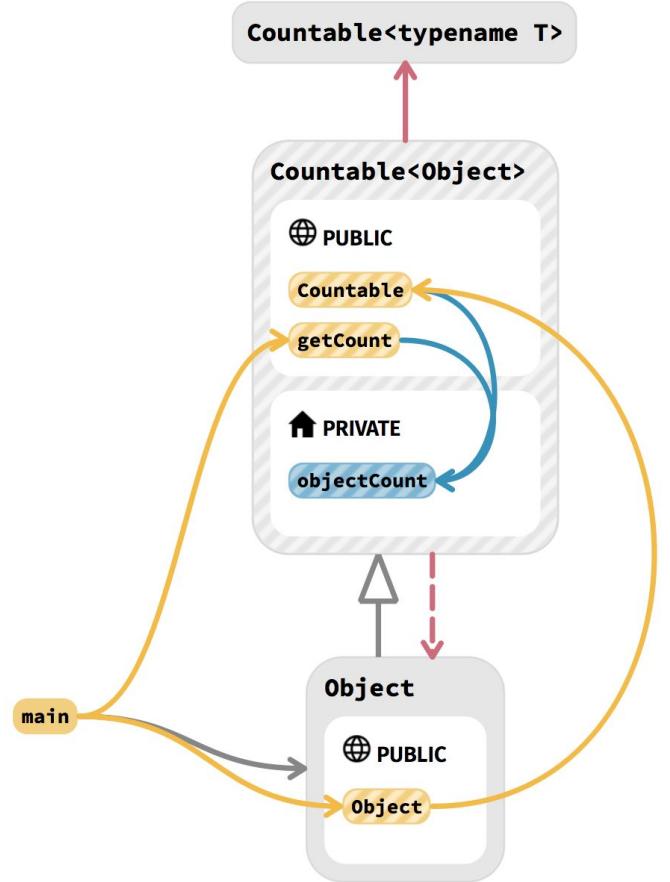
- Large amount of data
- Information spread across multiple locations/files
- Slow to process

```
1  template <typename T>
2  class Countable
3  {
4  public:
5      Countable() {
6          ++objectCount;
7      }
8
9
10     int getCount() const {
11         return objectCount;
12     }
13
14 private:
15     static int objectCount;
16 };
17
18 template <typename T> int Countable<T>::objectCount(0);
19
20 class Object : public Countable<Object>
21 {};
22
23 int main()
24 {
25     Object o;
26     int c = o.getCount();
27     return 0;
28 }
29
```



## Pros

- + Provides fast overview
- + Brings information together
- + Shows dependencies
- + Fast to process

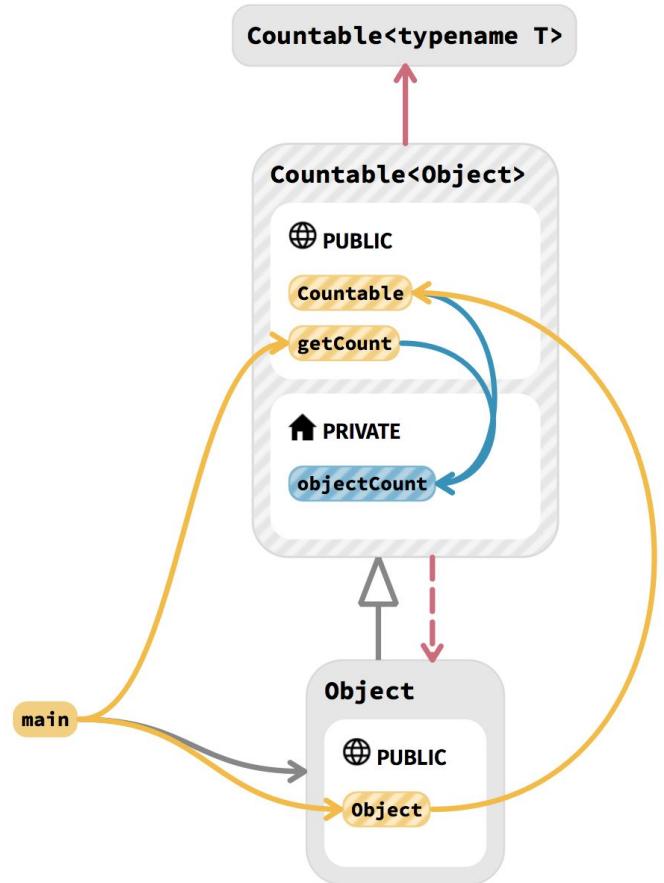


## Pros

- + Provides fast overview
- + Brings information together
- + Shows dependencies
- + Fast to process

## Cons

- Lacks a lot of information
- Can be cluttered
- Notation needs to be learned



```

1  template <typename T>
2  class Countable
3  {
4  public:
5      Countable() {
6          ++objectCount;
7      }
8
9
10     int getCount() const {
11         return objectCount;
12     }
13
14 private:
15     static int objectCount;
16 };
17
18 template <typename T> int Countable<T>::objectCount(0);
19
20 class Object : public Countable<Object>
21 {};
22
23 int main()
24 {
25     Object o;
26     int c = o.getCount();
27     return 0;
28 }
29
  
```

The screenshot shows the Sourcetrail IDE interface. At the top, there's a toolbar with various icons like back, forward, search, and file operations. Below the toolbar, a search bar displays the query `clang::FunctionProtoType::getNumParams`. To the right of the search bar are buttons for search, star, and filter. Further right are navigation arrows and a count of "126 references". On the far right are icons for export and help.

The main workspace has a large dark blue banner in the center with a white cat logo and the word "Sourcetrail" in large white letters. Below the banner, the number "5/5" is displayed. The rest of the screen shows code editor panes and toolbars. One visible toolbar on the left has a "5" button. A sidebar on the left lists files like `SemanticAnalysis.h`, `Type.h`, and `TypoCorrection.h`.

A code editor pane on the right shows a snippet of C++ code from `Type.h*` with 5 references:

```
3479     case EST_Unparsed:           return 0;
3480     case EST_Dynamic:            return getNumExceptions() * sizeof(QualType);
3481     case EST_ComputedNoexcept:   return 0;
3482     case EST_Uninstantiated:    return 0;
3483     case EST_Unevaluated:        return 0;
3484     case EST_Candidate:          return 0;
3485     default:                     return 0;
3486 }
3487
3488 static QualType getExceptionSpecDecl();
3489 static QualType getExceptionSpecTemplate();
3490 static QualType getNoexceptExpr();
```

Below this, another code editor pane shows a snippet from `TypoCorrection.h`:

```
3509     EPI.ExceptionSpec.Exceptions = exceptions();
3510 } else if (EPI.ExceptionSpec.Type == EST_ComputedNoexcept) {
3511     EPI.ExceptionSpec.NoexceptExpr = getNoexceptExpr();
3512 } else if (EPI.ExceptionSpec.Type == EST_Uninstantiated) {
3513     EPI.ExceptionSpec.SourceDecl = getExceptionSpecDecl();
3514     EPI.ExceptionSpec.SourceTemplate = getExceptionSpecTemplate();
3515 } else if (EPI.ExceptionSpec.Type == EST_Unevaluated) {
3516     EPI.ExceptionSpec.SourceDecl = getExceptionSpecDecl();
```



# LLVM/Clang LibTooling

```
1 class Object
2 {
3     public:
4         int getValue() const
5         {
6             return m_value;
7         }
8
9         void setValue(int value)
10        {
11            m_value = value;
12        }
13
14     private:
15         int m_value = 0;
16 };
17
18
```

```
TranslationUnitDecl <><invalid sloc>> <><invalid sloc>>
`-CXXRecordDecl <test.cpp:2:1, line:17:1> line:2:7 class Object definition
|-CXXRecordDecl <col:1, col:> col:7 implicit class Object
|-AccessSpecDecl <line:4:1, col:7> col:1 public
|-CXXMethodDecl <line:5:2, line:8:2> line:5:6 getValue 'int (void) const'
`-CompoundStmt <line:6:2, line:8:2>
    `-ReturnStmt <line:7:3, col:10>
        `-ImplicitCastExpr <col:10> 'int' <LValueToRValue>
            `MemberExpr <col:10> 'const int' lvalue ->m_value
            `CXXThisExpr <col:10> 'const class Object *' this
|-CXXMethodDecl <line:10:2, line:13:2> line:10:7 setValue 'void (int)'
|-ParmVarDecl <col:16, col:20> col:20 used value 'int'
`-CompoundStmt <line:11:2, line:13:2>
    `-BinaryOperator <line:12:3, col:13> 'int' lvalue '='
        |-MemberExpr <col:3> 'int' lvalue ->m_value
        |`CXXThisExpr <col:3> 'class Object *' this
        |-ImplicitCastExpr <col:13> 'int' <LValueToRValue>
            `DeclRefExpr <col:13> 'int' lvalue ParmVar 'value' 'int'
|-AccessSpecDecl <line:15:1, col:8> col:1 private
|-FieldDecl <line:16:2, col:16> col:6 referenced m_value 'int'
`-IntegerLiteral <col:16> 'int' 0
```

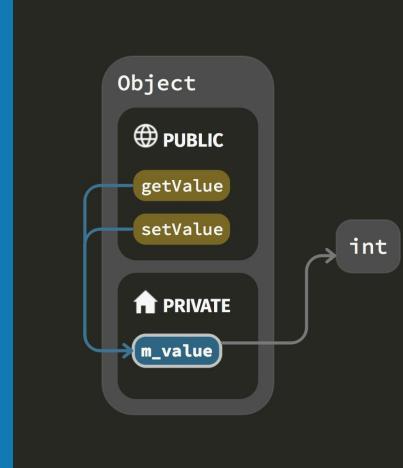
CODE

->

AST

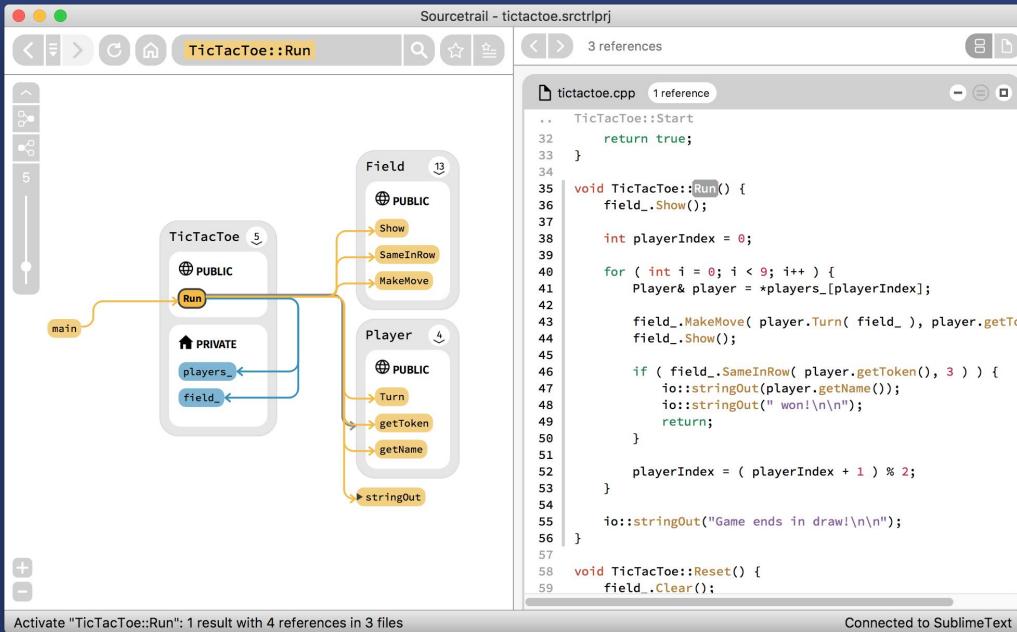
->

GRAPH





graph



<https://sourcetrail.com>

search

code



## Closing Comments

Visualization is a great tool if it supports your use-case.

Challenges to build good visualizations:

- Collect accurate data
- Design representative visualization
- Provide intuitive user interaction

# Thank you! Questions?



<https://sourcetrail.com>

[mail@sourcetrail.com](mailto:mail@sourcetrail.com)  
Twitter: @Sourcetrail

**Eberhard Gräther**  
@egraether  
egraether@coati.io

**Coati Software KG**  
Jakob-Haringer-Straße 1/127  
5020 Salzburg  
Austria

© Coati Software KG