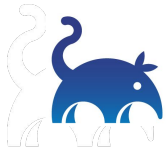
The Untapped Potential of Software Visualization

by Eberhard Gräther

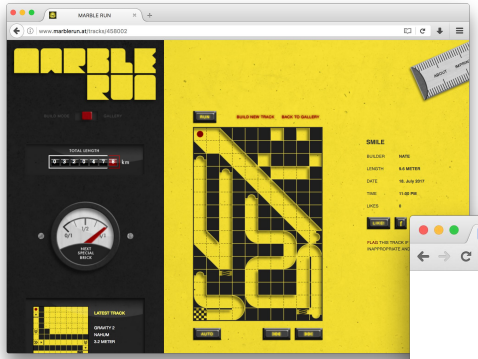
C++Now 2018

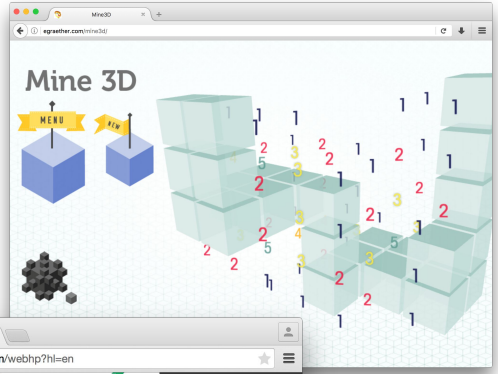
Introduction

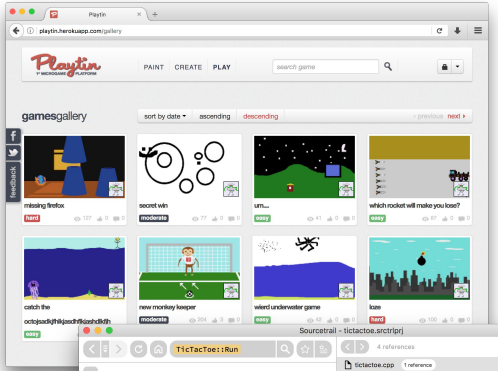
0/5

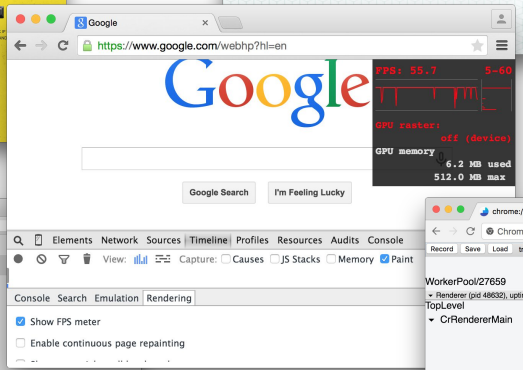


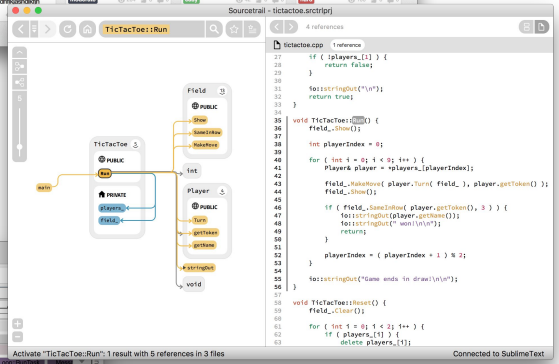
Work

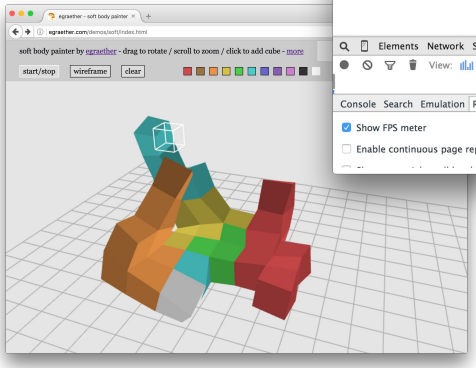

MARBLE RUN

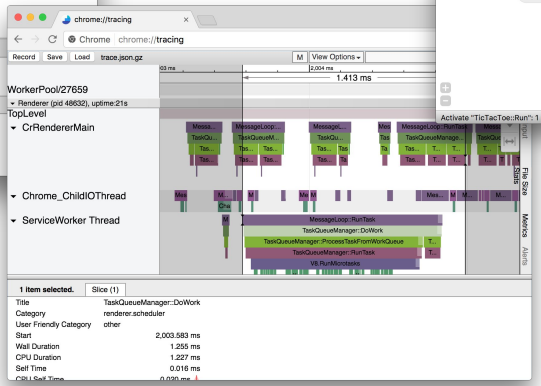

Mine 3D
MENU
PLAY


Playtin
PAINT
CREATE
PLAY
gamesgallery


Google
https://www.google.com/webhp?hl=en
Google
Google Search
I'm Feeling Lucky
FPS: 55.7
GPU raster:
GPU memory


Sourcetrail - tictactoe.srctrlprj
TicTacToe::Run


soft body painter by egraether - drag to rotate / scroll to zoom / click to add cube - more
start/stop
wireframe
clear


chrome://tracing
Record
Save
Load
WorkerPool/27659
CrRendererMain
Chrome_ChildIOThread
ServiceWorker Thread

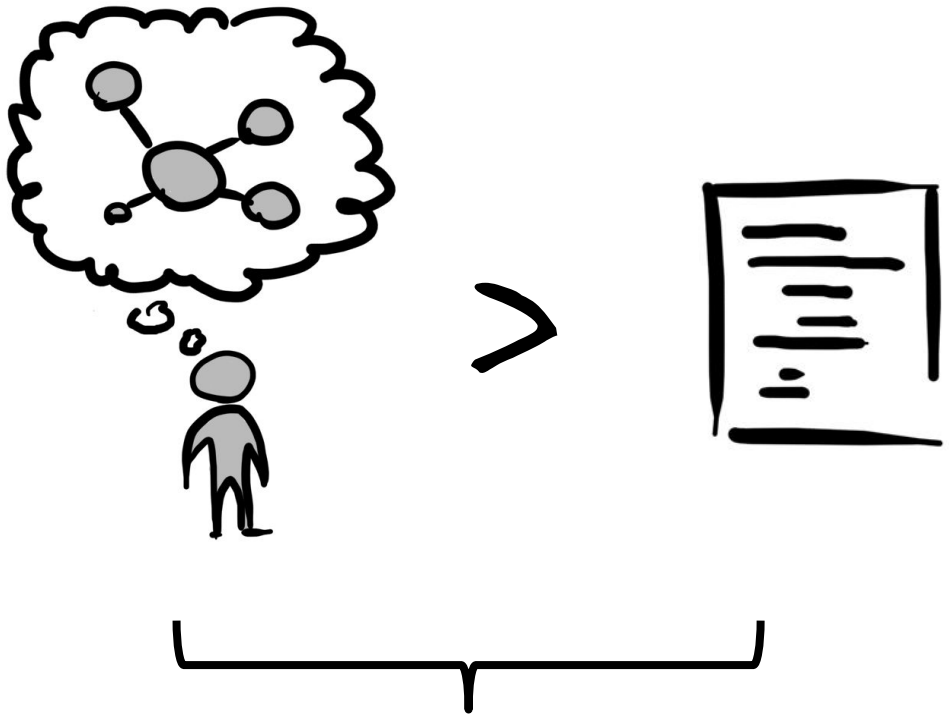
“Programmers tend to adapt to the level of representation ***provided by the computer****, instead of adapting the computers representations to their* ***perceptive abilities****.”*

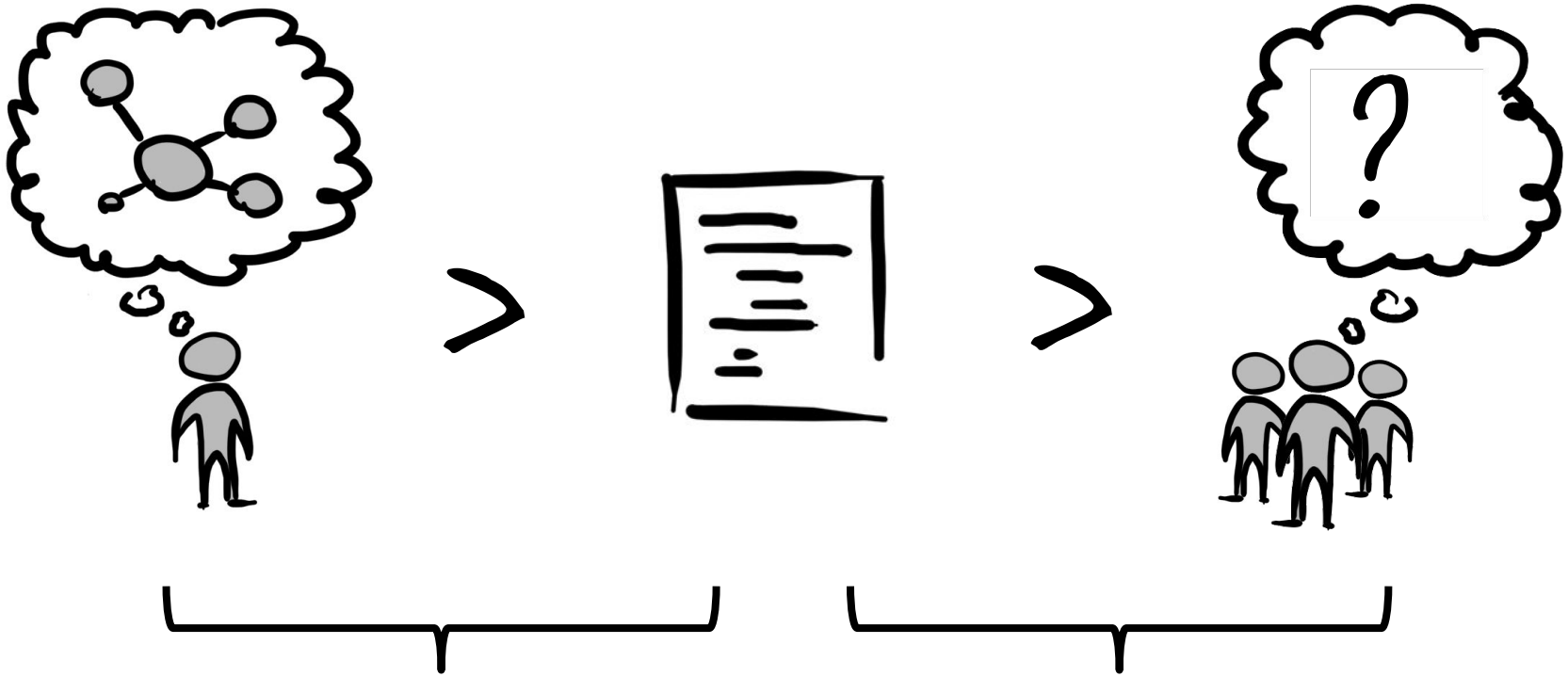
- Stephan Dhiel

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

Decoding Code

1/5

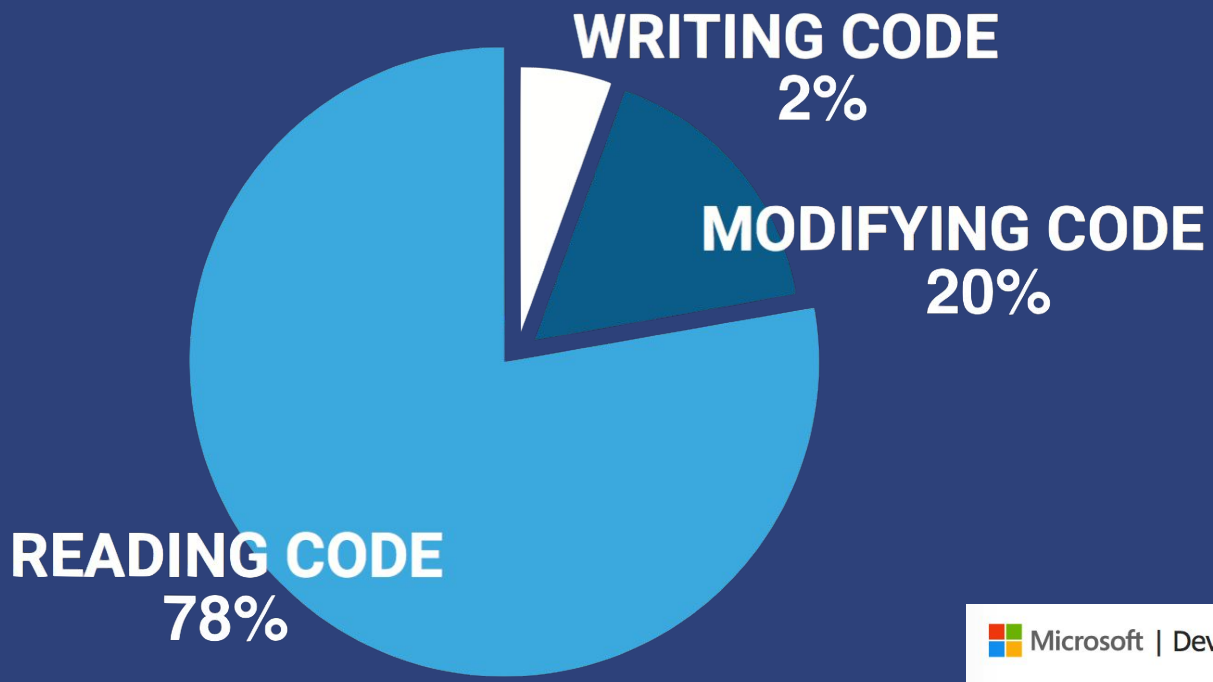

WRITING


?
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

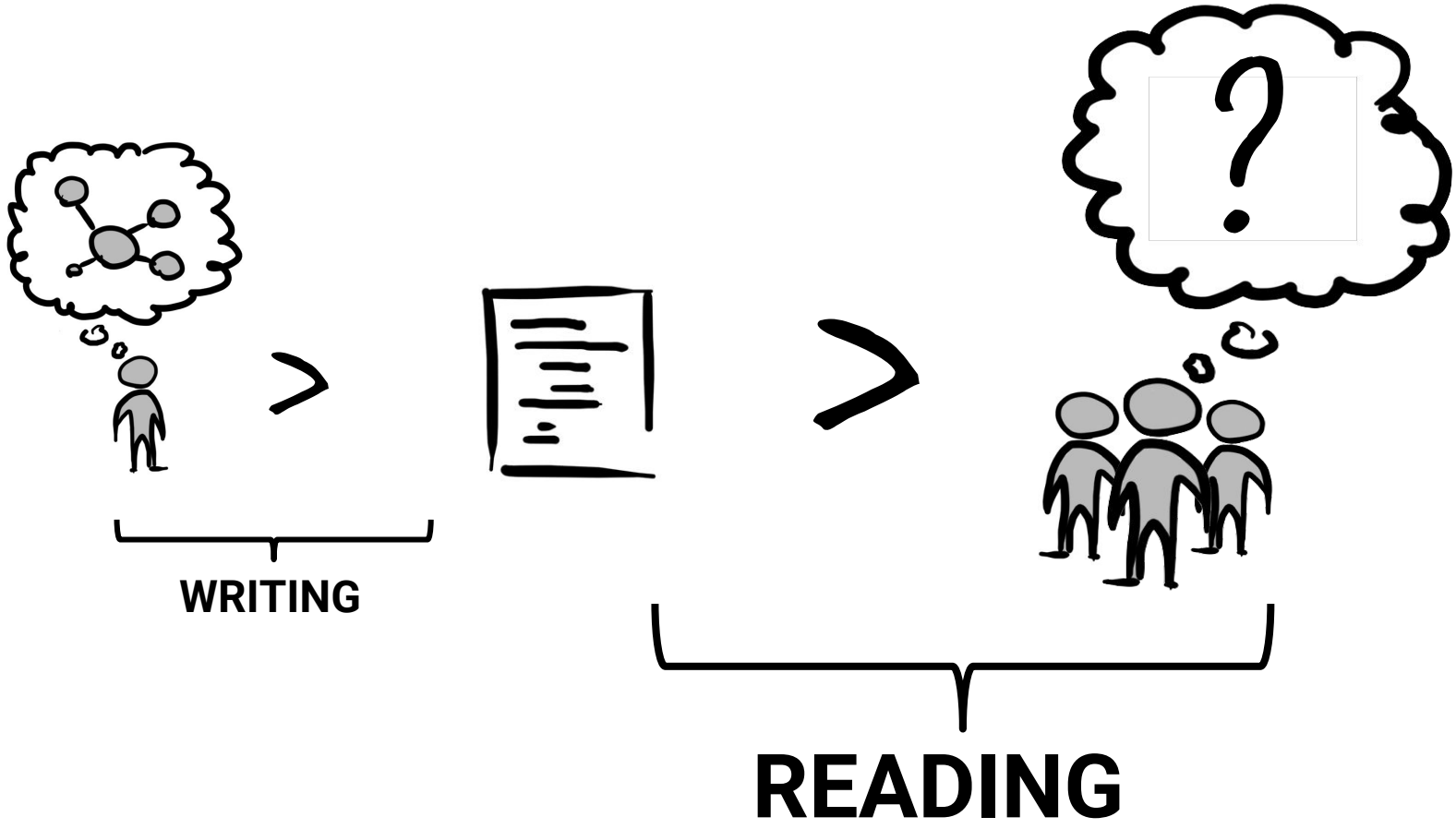
(Clean Code: A Handbook of Agile Software Craftsmanship)



Microsoft | Developer

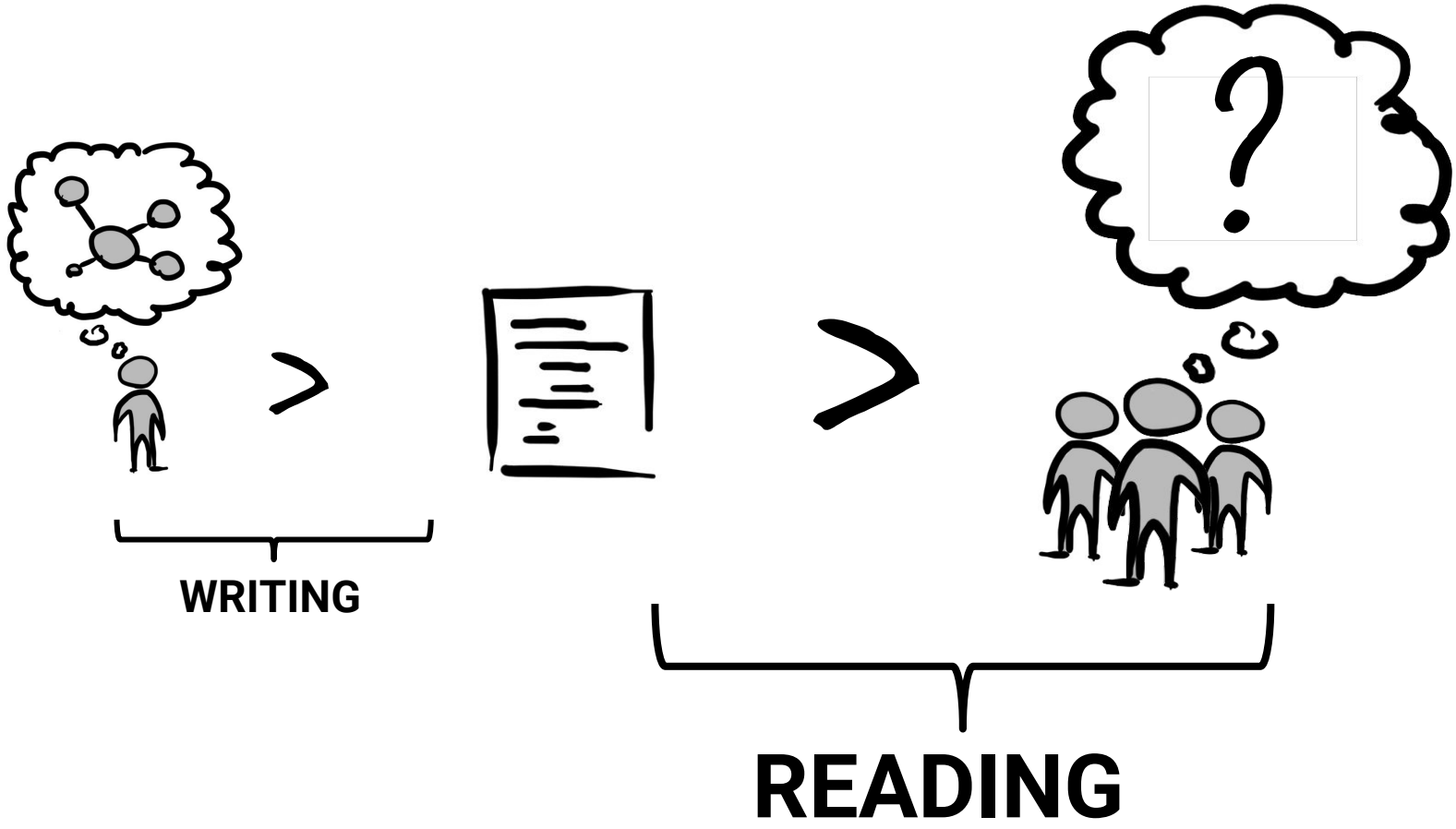
Peter Hallam's WebLog

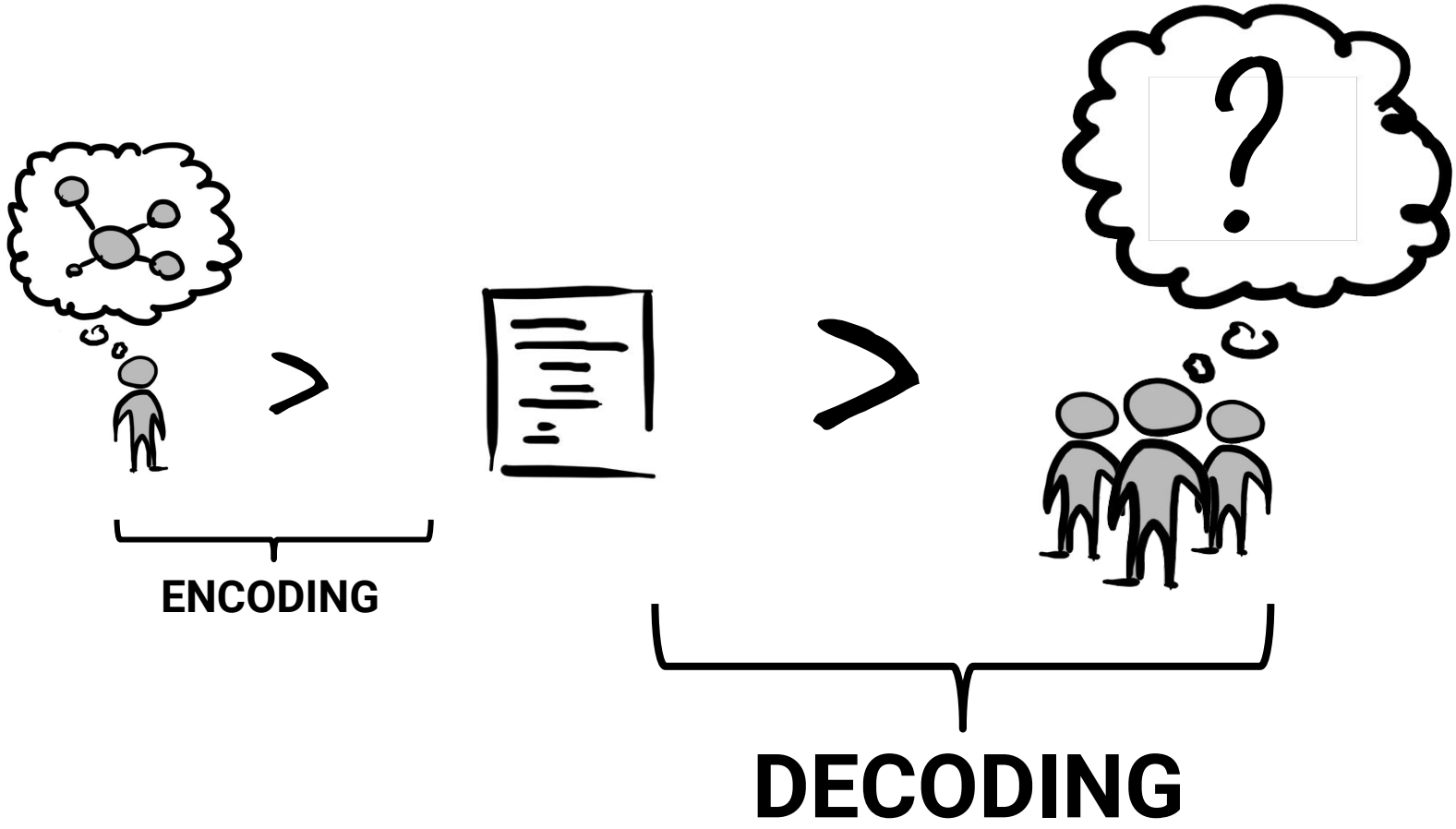
* https://blogs.msdn.microsoft.com/peterhal/2006/01/04/what-do-programmers-really-do-anyway-aka-part-2-of-the-yardstick-saga/


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


WRITING
READING


?
ENCODING
DECODING

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

```
1
2  template <typename T>
3  class Countable
4  {
5  public:
6      Countable() {
7          ++objectCount;
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
2  template <typename T>
3  class Countable
4  {
5  public:
6      Countable() {
7          ++objectCount;
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
```

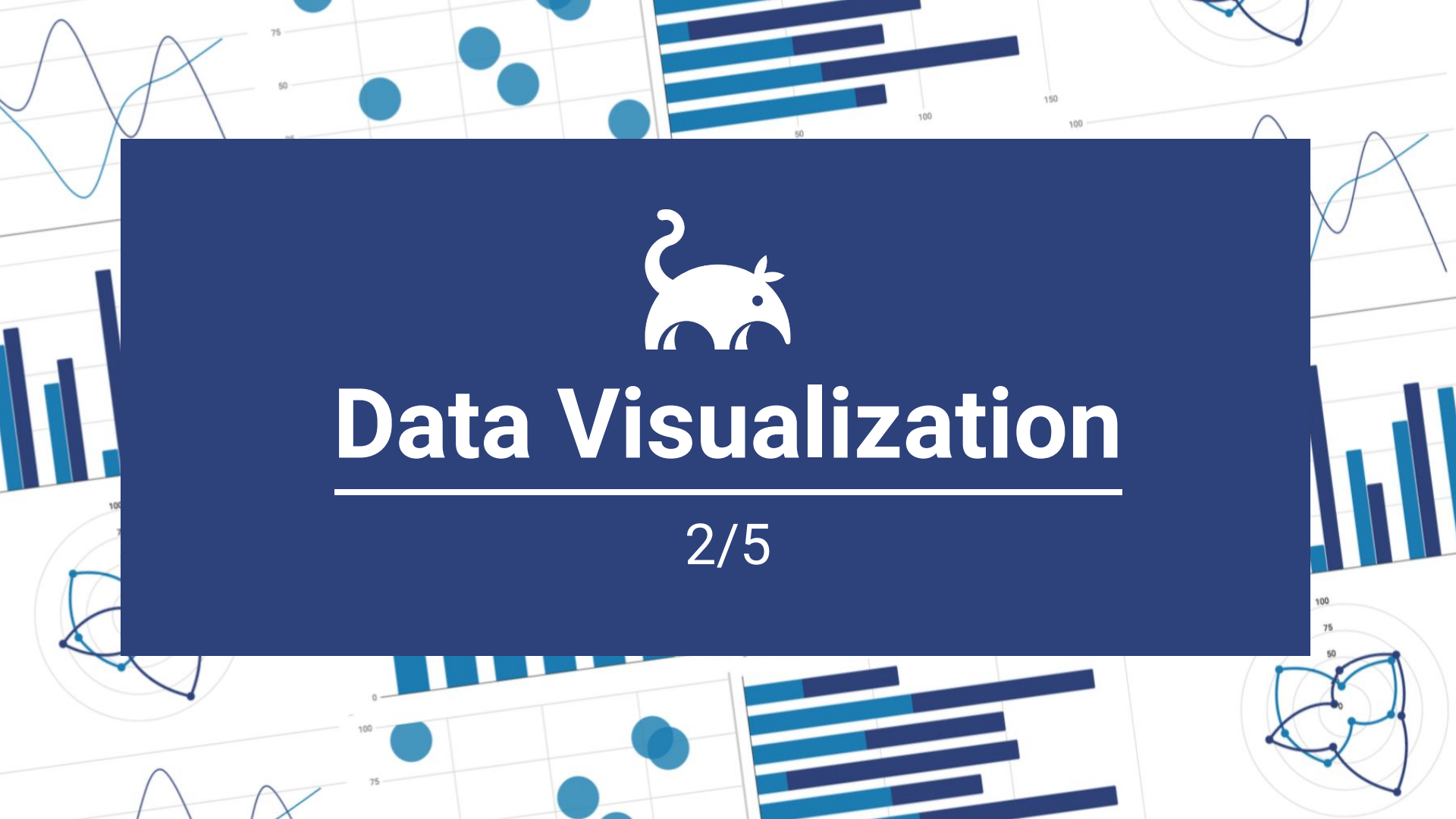
C++ source code is textual data with high information density.

For readability:

- **syntax highlighting**
- **naming convention**
- **whitespace**

But not ideal for our mind!

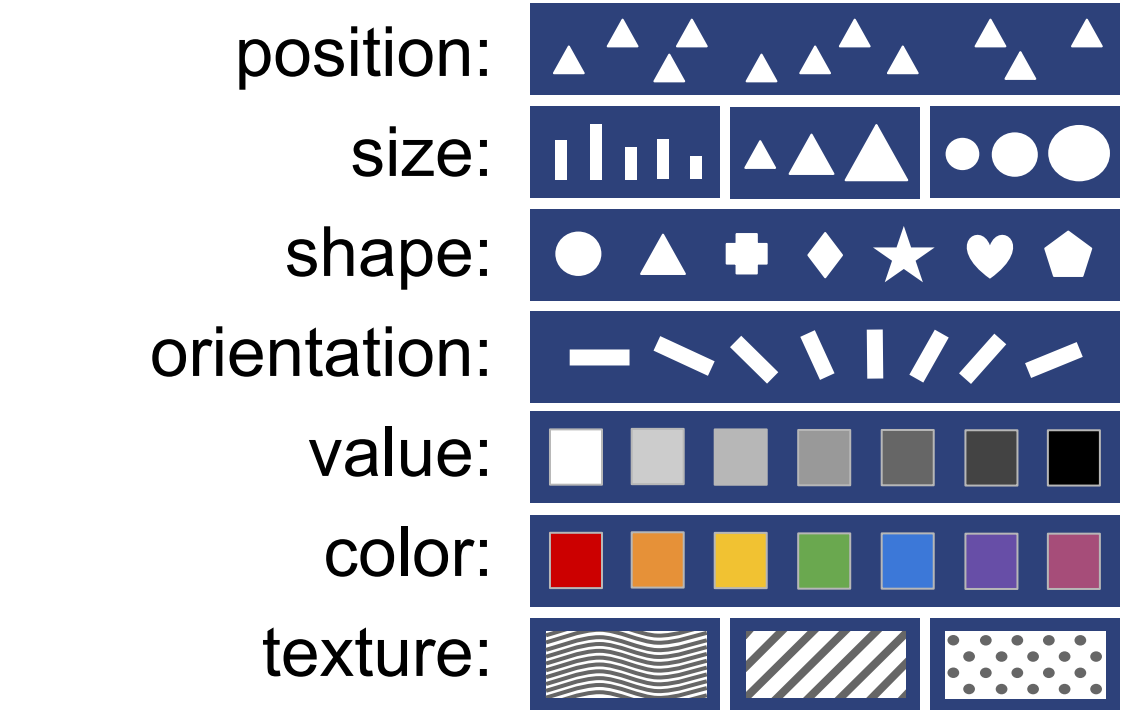
```
1
2  template <typename T>
3  class Countable
4  {
5  public:
6      Countable() {
7          ++objectCount;
8      }
9
10     int getCount() const {
11         return objectCount;
12     }
13
14 private:
15     static int objectCount;
16 };
17
18 template <typename T> int Countable<T>::objectCount(0);
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
```



Data Visualization

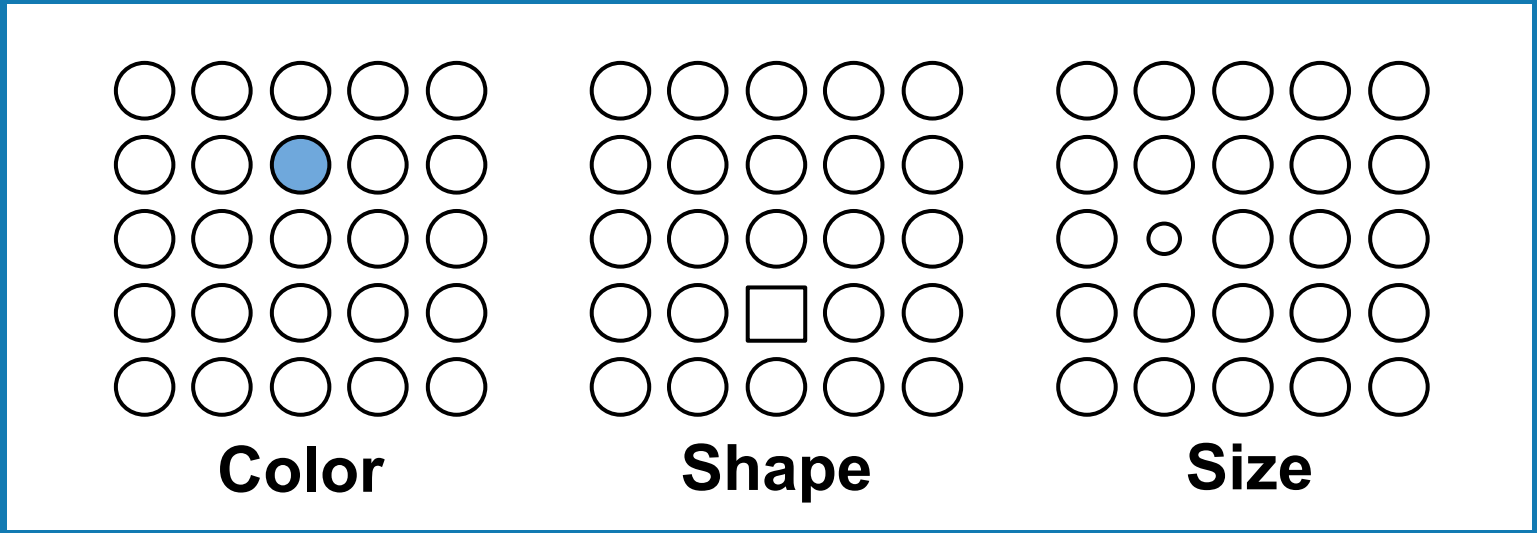
2/5

Visual Variables



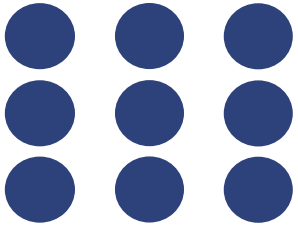
Preattentive Perception

Features are perceived within 200 ms:

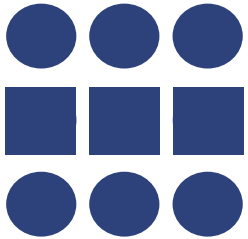


(Software Visualization, Dhiel 2007, S. 18)

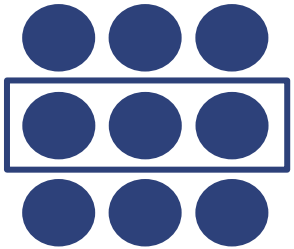
Principles of Grouping



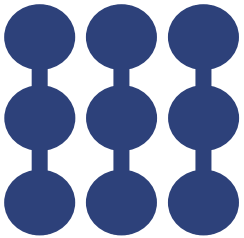
Proximity



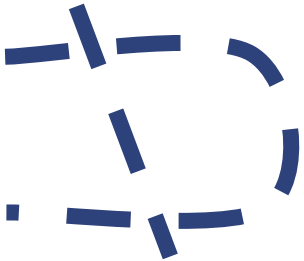
Similarity



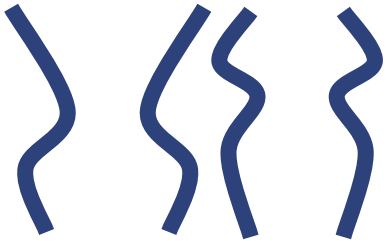
Enclosure



Connection



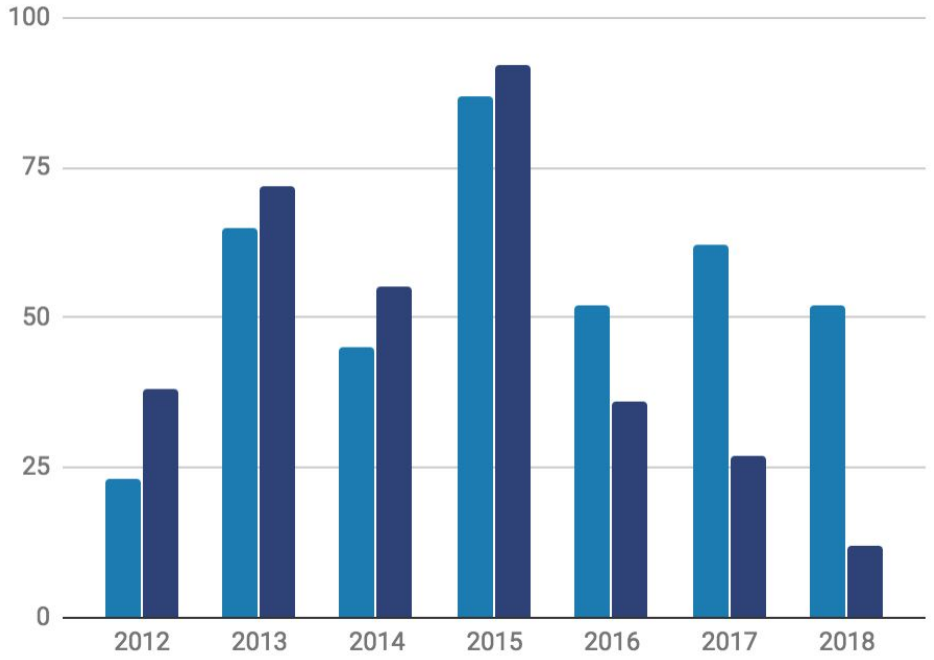
Continuation



Symmetry

Sample 1

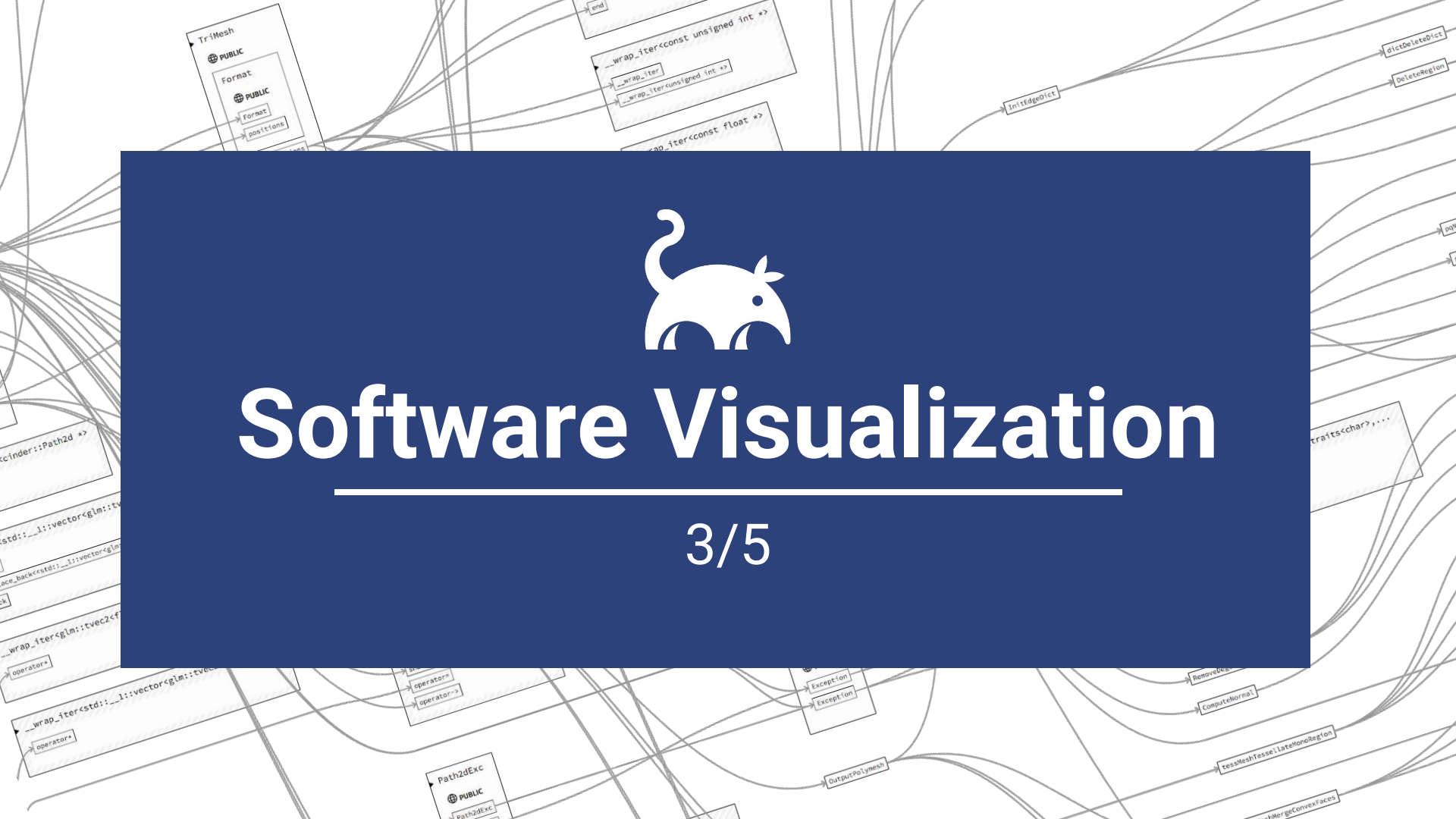
2012	23	38
2013	65	72
2014	45	55
2015	87	92
2016	52	36
2017	62	27
2018	52	12



Sample 2

```
template<typename T>class Countable{
public:Countable(){++objectCount;}int
getCount()const{return objectCount;}
private:static int objectCount;};
template<typename T>int Countable<T>::
objectCount(0);class Object:public
Countable<Object>{};int main(){Object
o;int c=o.getCount();return 0;}
```

```
template <typename T>
class Countable
{
public:
    Countable() {
        ++objectCount;
    }

    int getCount() const {
        return objectCount;
    }

private:
    static int objectCount;
};

template <typename T> int Countable<T>::objectCount(0);

class Object : public Countable<Object>
{};

int main()
{
    Object o;
    int c = o.getCount();
    return 0;
}
```

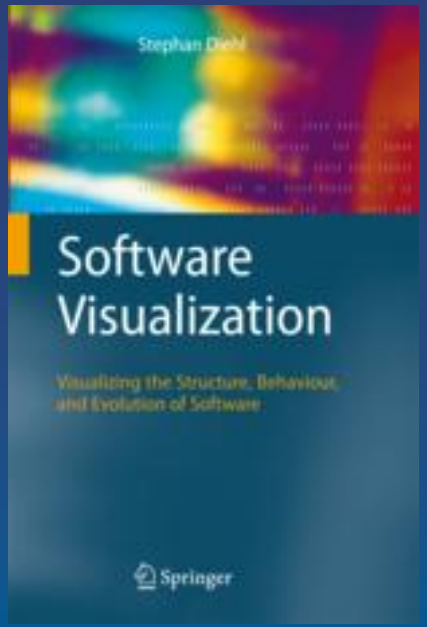


Software Visualization

3/5

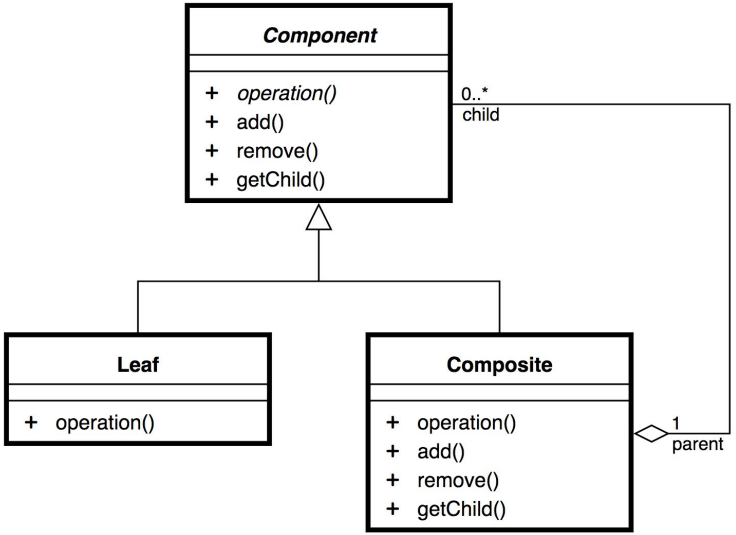
Software Visualization

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



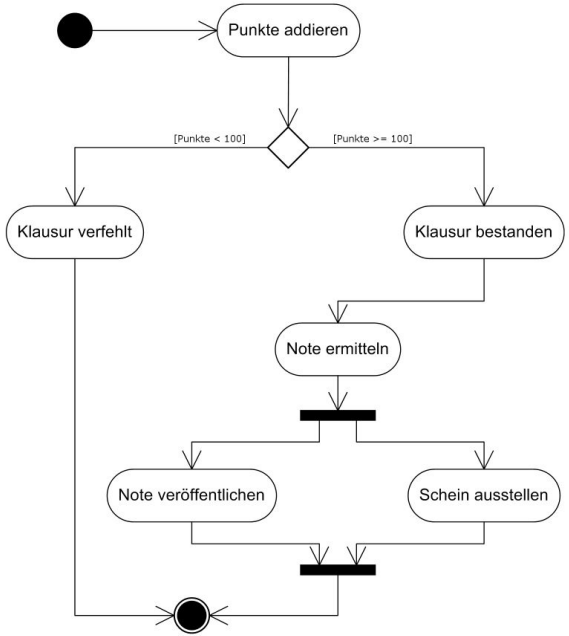
Software Visualization, Stephan Dhiel, 2007
https://www.springer.com/us/book/9783540465041

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)



Activity Diagram

By Original PNG Aktivity 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

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
CppDepend Code Metrics Treemap: https://www.cppdepend.com/Doc_Treemap

Behaviour Examples

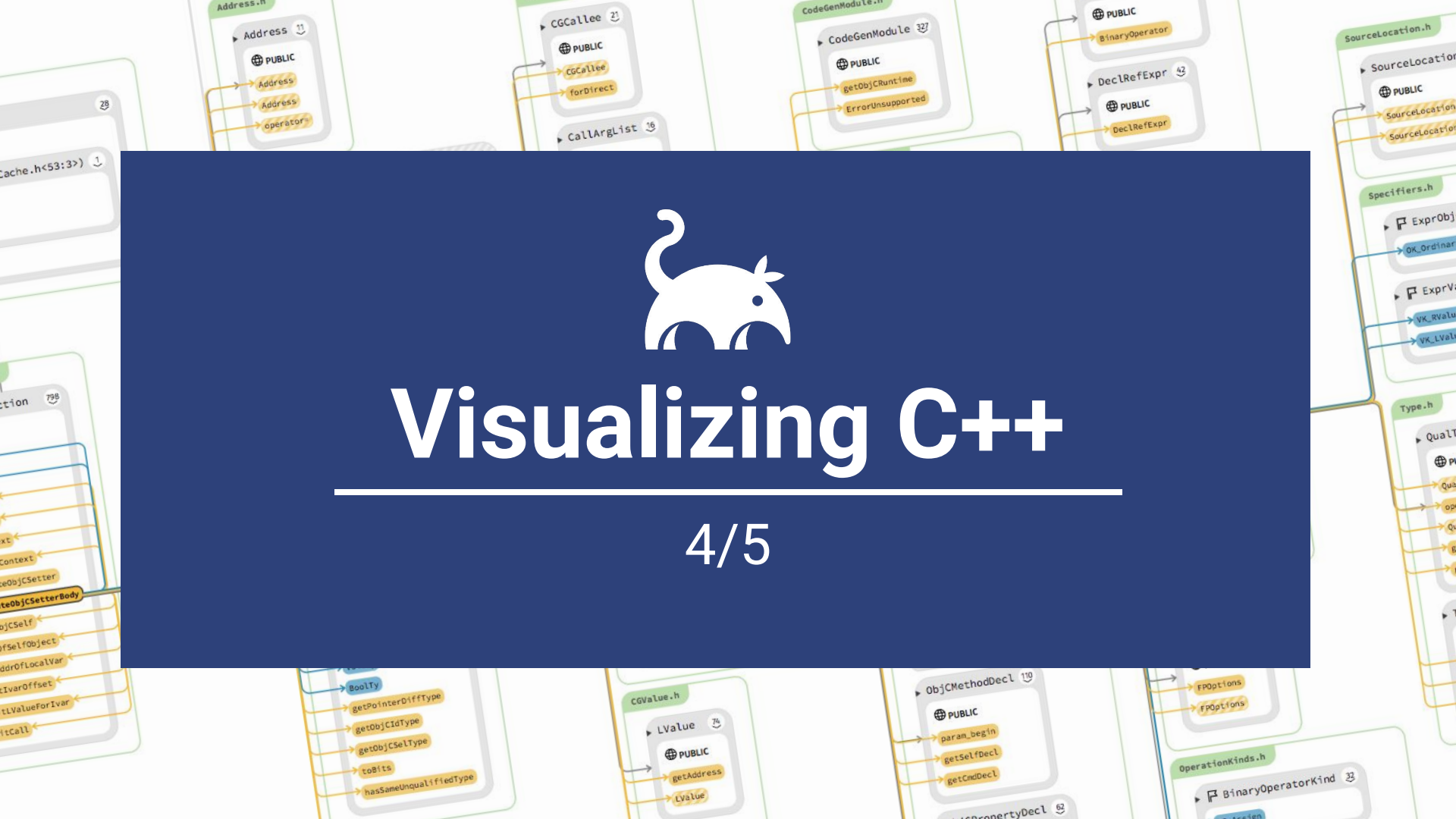
VisuAlgo Sorting Algorithms: https://visualgo.net/en/sorting

Chrome Tracing Flame Chart: chrome://tracing/

Evolution Examples

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

CppDepend Trend Monitoring: https://www.cppdepend.com/Doc_Trend



Visualizing C++

4/5

Goal:

Overview of existing symbols

Dependencies between symbols

```
1
2  template <typename T>
3  class Countable
4  {
5  public:
6      Countable() {
7          ++objectCount;
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
2  template <typename T>
3  class Countable
4  {
5  public:
6      Countable() {
7          ++objectCount;
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
```

Nodes

Type

```
class Type { };
```

function

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

variable

```
int variable;
```

Class

PUBLIC

method

PRIVATE

field

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

Edges

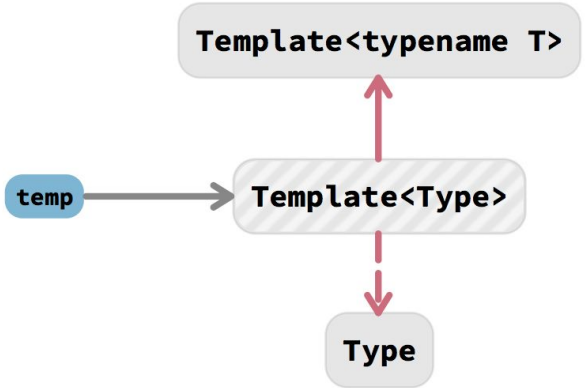

```
void function() { variable = 1; }
```


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

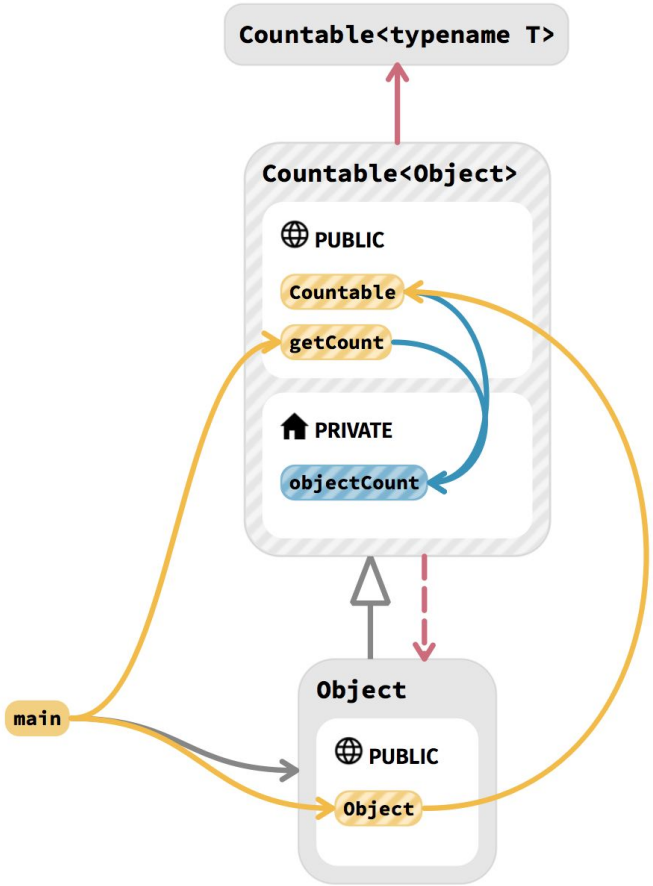

```
void function() { Type t; }
```

Edges


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




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




```
1
2   template <typename T>
3   class Countable
4   {
5   public:
6       Countable() {
7           ++objectCount;
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
2   template <typename T>
3   class Countable
4   {
5   public:
6       Countable() {
7           ++objectCount;
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
```

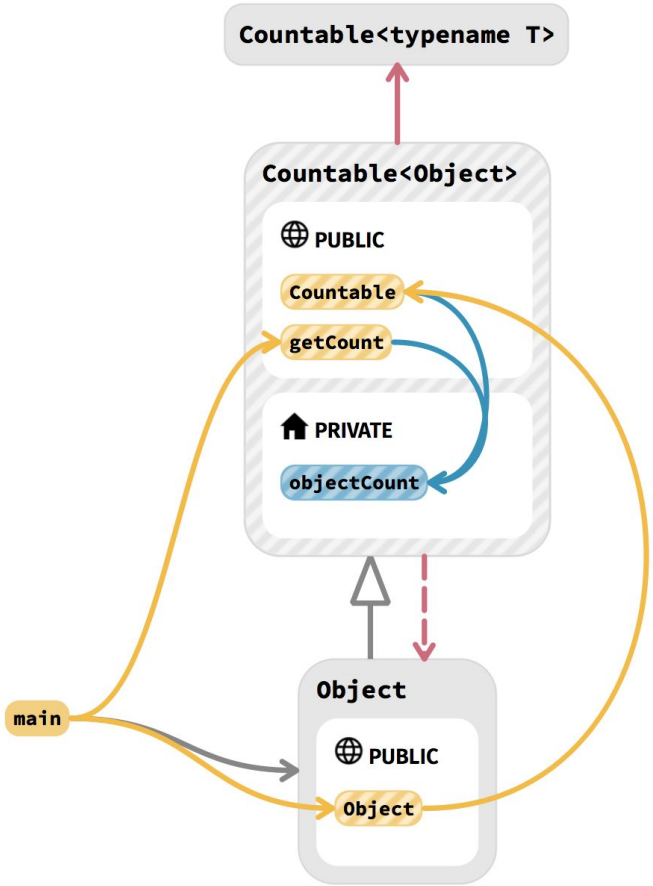
Pros

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

Cons

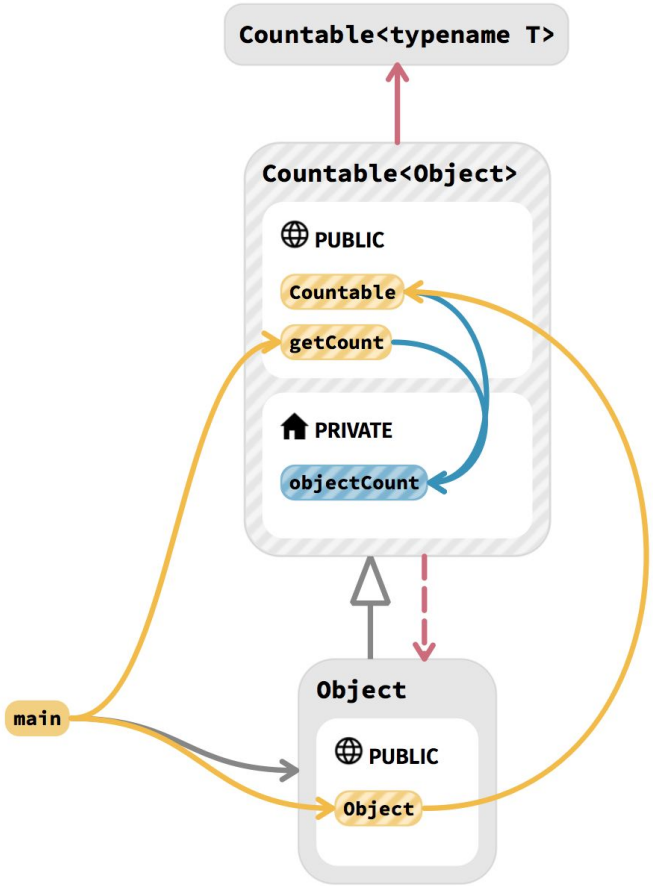
– Large amount of data
– Information spread across multiple locations/files
– Slow to process

```
1
2   template <typename T>
3   class Countable
4   {
5   public:
6       Countable() {
7           ++objectCount;
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

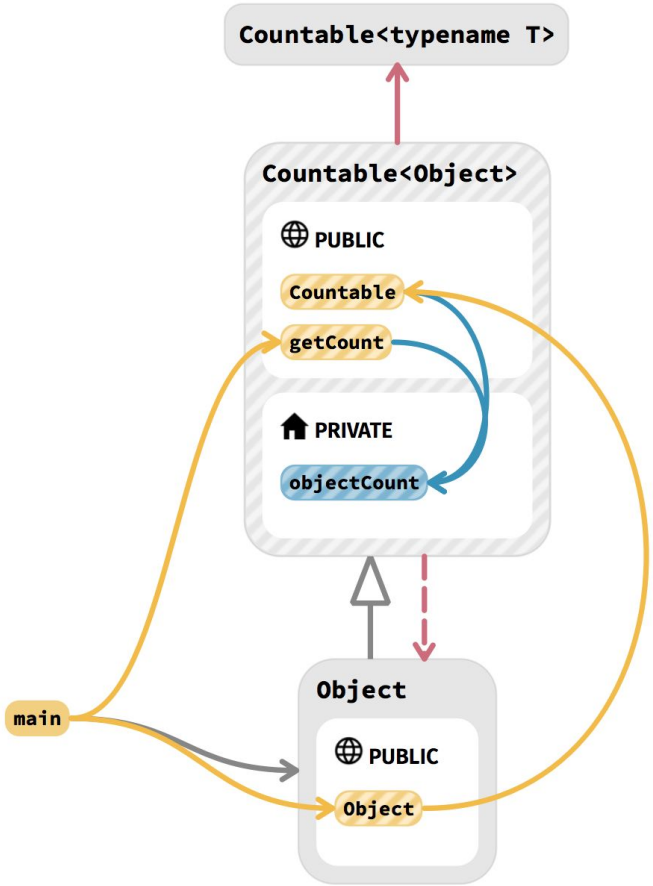


Pros

+ 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



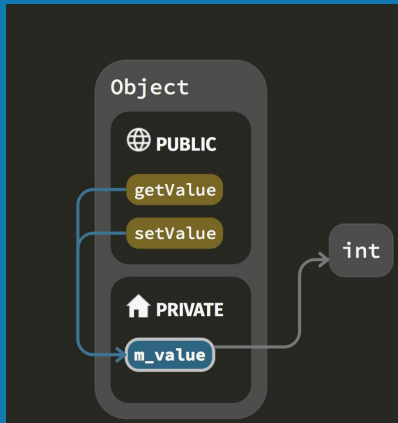

```
1
2   template <typename T>
3   class Countable
4   {
5   public:
6       Countable() {
7           ++objectCount;
8       }
9
10      int getCount() const {
11          return objectCount;
12      }
13
14  private:
15      static int objectCount;
16  };
17
18  template <typename T> int Countable<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
```


Sourcetrail - Clang_6_0_0.srctrlprj
clang::FunctionProtoType::getNumParams
126 references
Type.h*
5 references
case EST_Unparsed: return 0;
case EST_Dynamic: return getNumExceptions() * sizeof(QualTyp
EPI.ExceptionSpec.Exceptions = exceptions();
} else if (EPI.ExceptionSpec.Type == EST_ComputedNoexcept) {
EPI.ExceptionSpec.NoexceptExpr = getNoexceptExpr();
} else if (EPI.ExceptionSpec.Type == EST_Uninstantiated) {
EPI.ExceptionSpec.SourceDecl = getExceptionSpecDecl();
EPI.ExceptionSpec.SourceTemplate = getExceptionSpecTemplate();
} else if (EPI.ExceptionSpec.Type == EST_Unevaluated) {
TypoCorrection.h
FunctionCallFilterCCC
PUBLIC
Sourcetrail
5/5

LLVM/Clang LibTooling

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

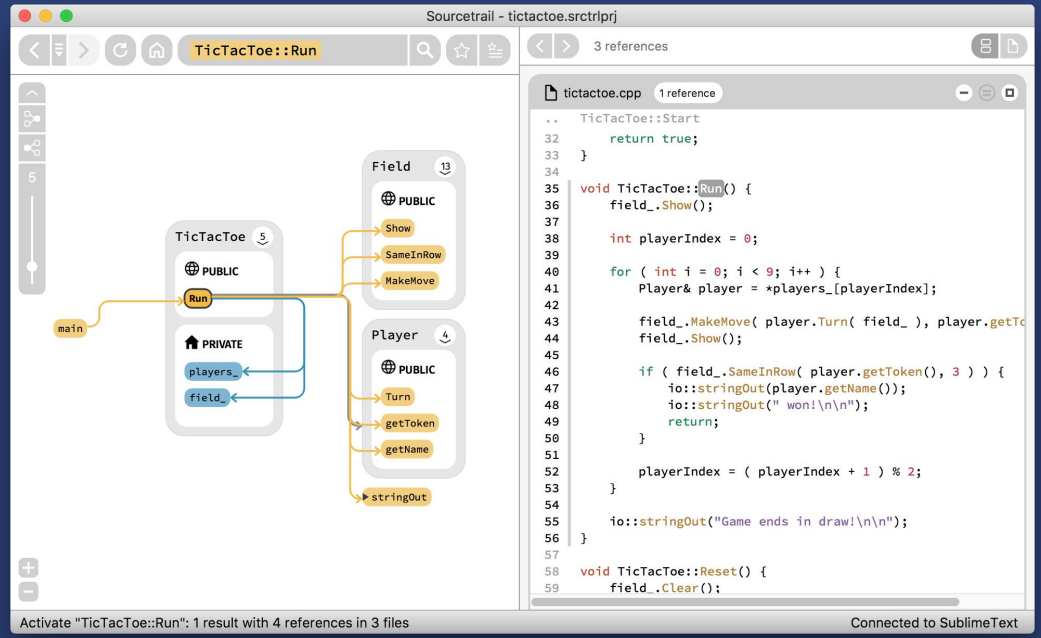
```
TranslationUnitDecl <<invalid sloc>> <invalid sloc>
`-CXXRecordDecl <test.cpp:2:1, line:17:1> line:2:7 class Object definition
|-CXXRecordDecl <col:1, col:7> col:7 implicit class Object
|-AccessSpecDecl <line:4:1, col:7> col:1 public
|-CXXMethodDecl <line:5:2, line:8:2> line:5:6 getValue 'int (void) 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

SOURCETRAIL

search



code

graph

https://sourcetrail.com

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
Twitter: @Sourcetrail

Eberhard Gräther
@egraether
egraether@coati.io

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