10 TECHNIQUES TO UNDERSTAND EXISTING CODE

Jonathan Boccara

@JoBoccara

Fluent {C++}

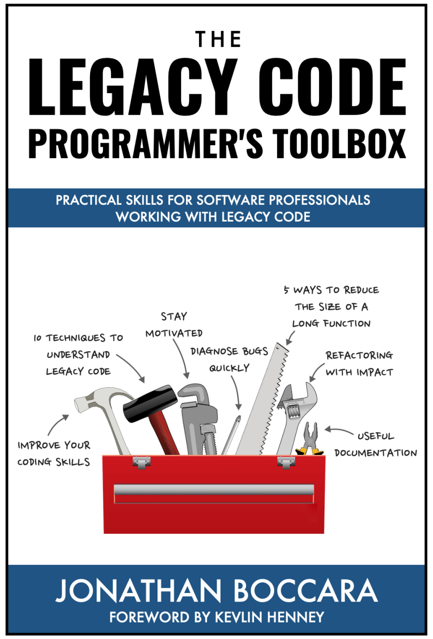
MUREX™
FINANCIAL SOFTWARE

10 TECHNIQUES TO UNDERSTAND EXISTING CODE

Hi, I'm Jonathan Boccara!

@JoBoccara

Fluent {C++}
EXPRESSIVE CODE IN C++


THE
LEGACY CODE
PROGRAMMER'S TOOLBOX
PRACTICAL SKILLS FOR SOFTWARE PROFESSIONALS WORKING WITH LEGACY CODE
5 ways to reduce the size of a long function
Stay motivated
10 techniques to understand legacy code
Diagnose bugs quickly
Refactoring with impact
Improve your coding skills
Useful documentation
JONATHAN BOCCARA
FOREWORD BY KEVLIN HENNEY

MUREX™
FINANCIAL SOFTWARE

10 TECHNIQUES TO UNDERSTAND EXISTING CODE

10 TECHNIQUES TO UNDERSTAND EXISTING CODE

OUTLINE:

Exploring the code (3 techniques)

Becoming a code speed-reader (4 techniques)

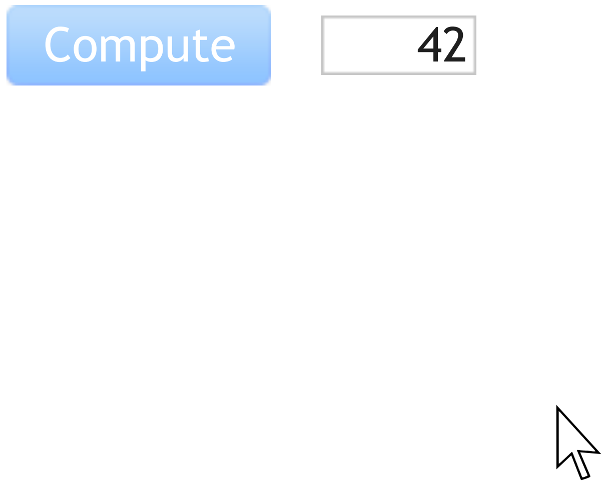
Understanding code in details (3 techniques)

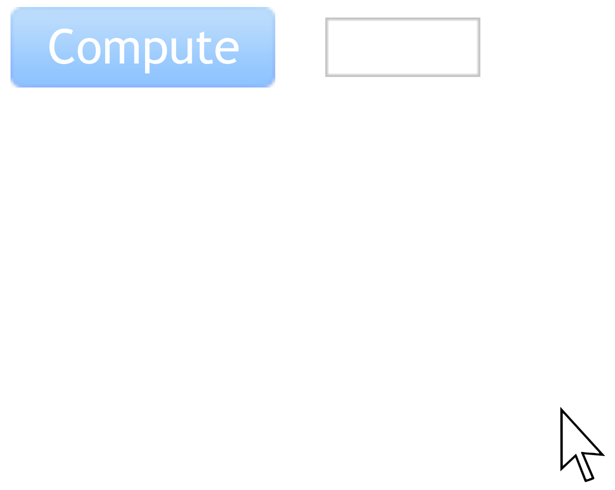
Exploring the code

(3 techniques)

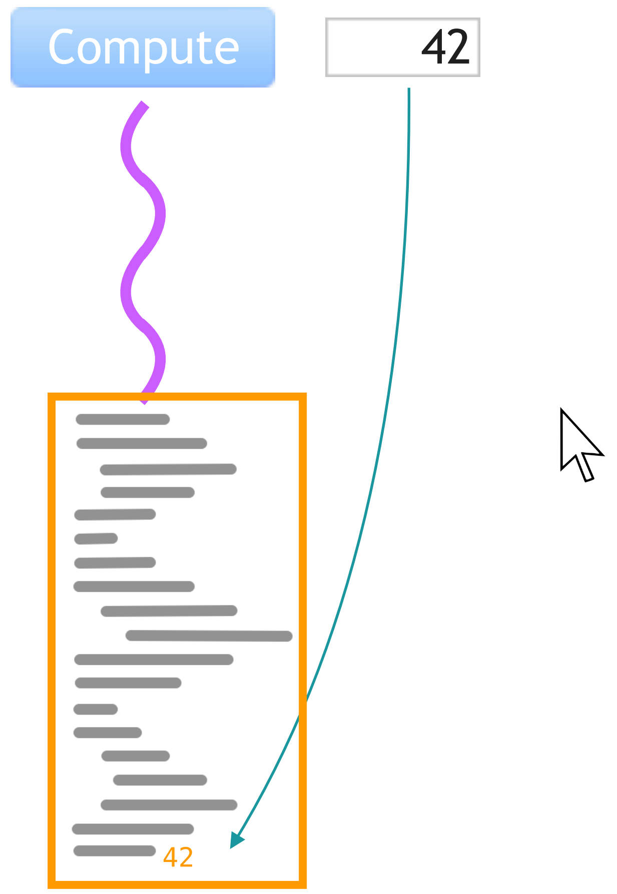
TECHNIQUE #1

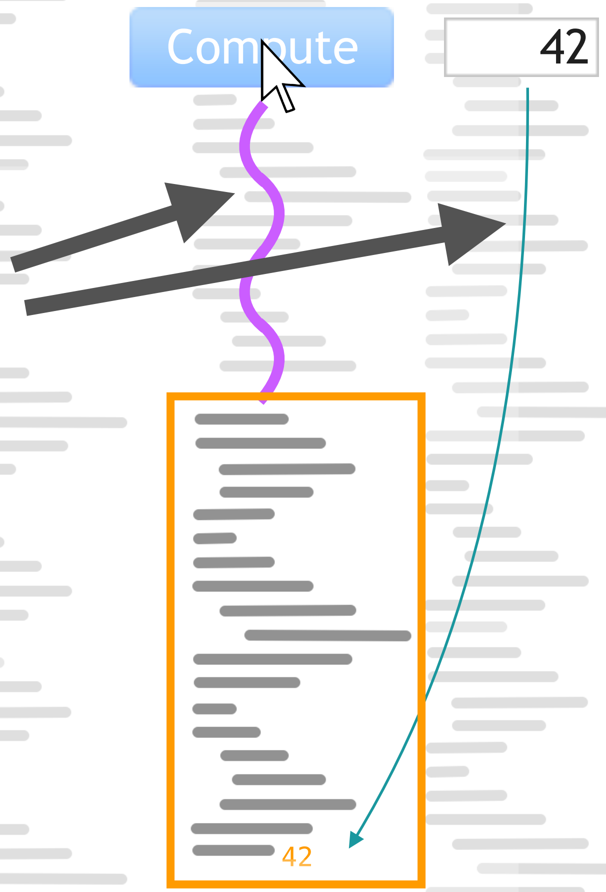
Know your I/O frameworks.


Compute
42


Compute

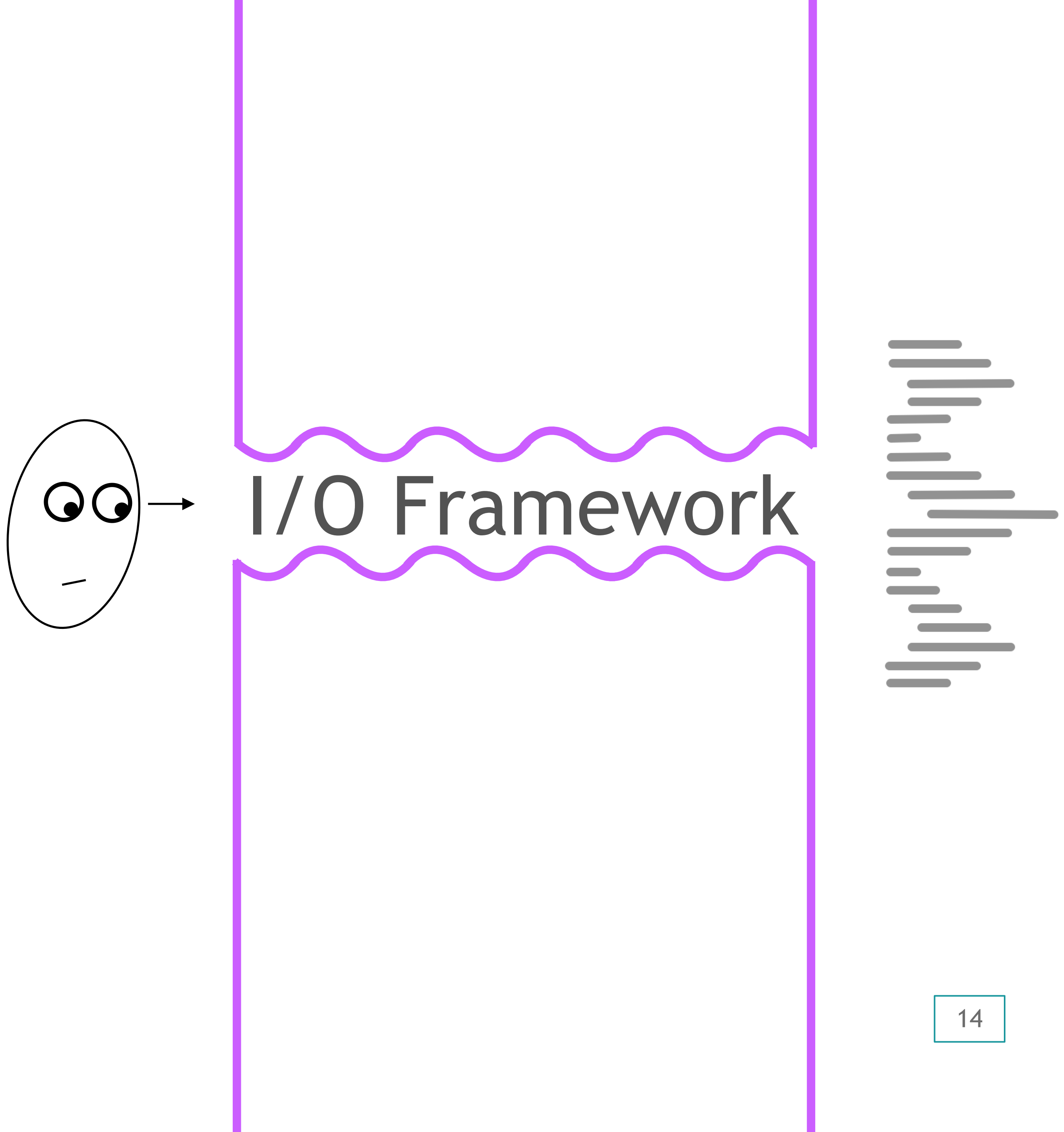
Compute


Compute
42
42



Get familiar with the code of your UI framework

UI Framework
REST handler
Logs
Unit tests



TECHNIQUE #2

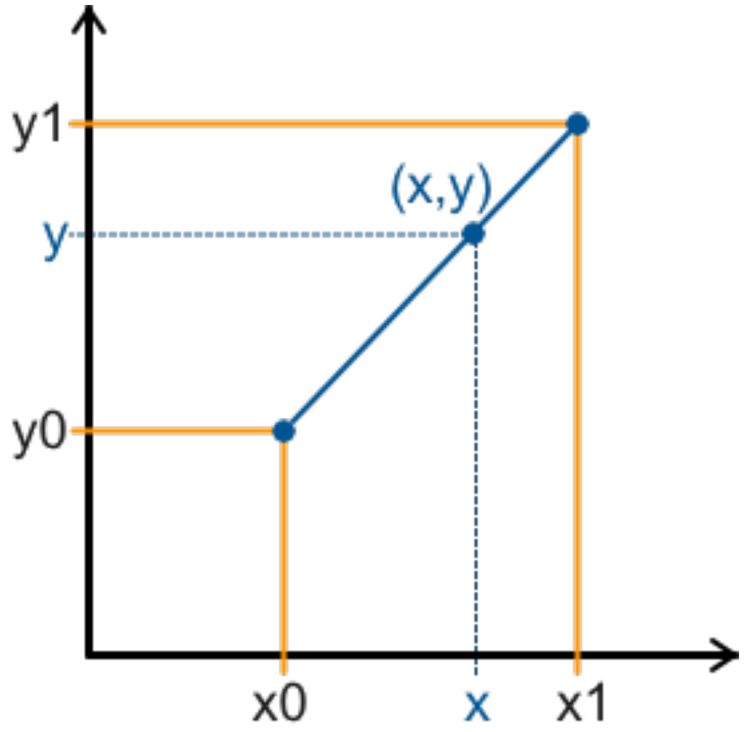
Find a stronghold.

WTF

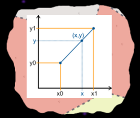
map = codebase

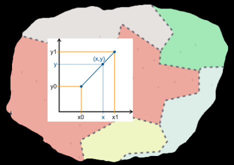
stronghold = code you understand perfectly well
even one line

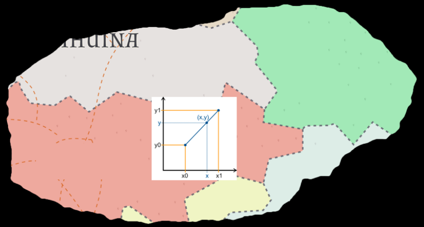
Chances are you can figure out the immediate surroundings too.



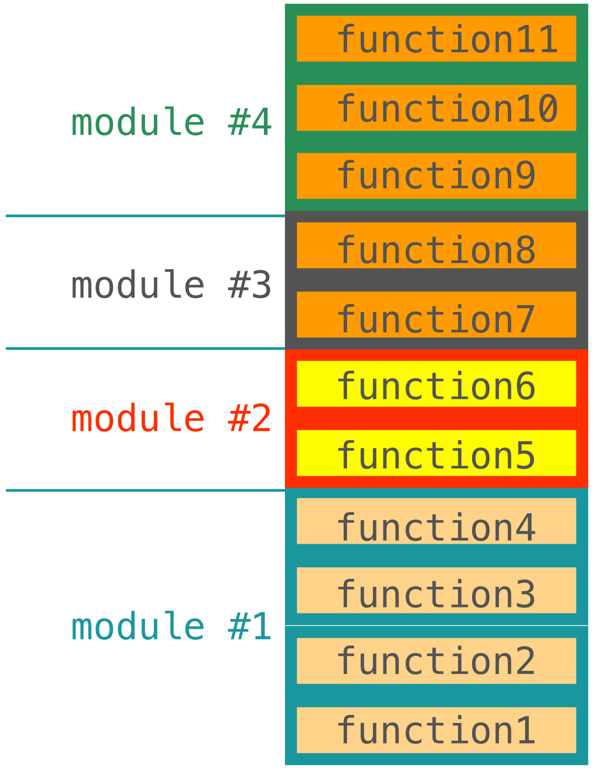
$$y = y0 + (x - x0) \times \frac{y1 - y0}{x1 - x0}$$


y1
(x,y)
y
y0
x0
x
x1

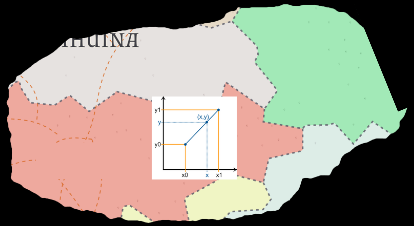

y1
y
y0
(x,y)
x0
x
x1

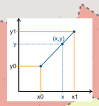

IHUINA
y1
(x,y)
y
y0
x0
x
x1

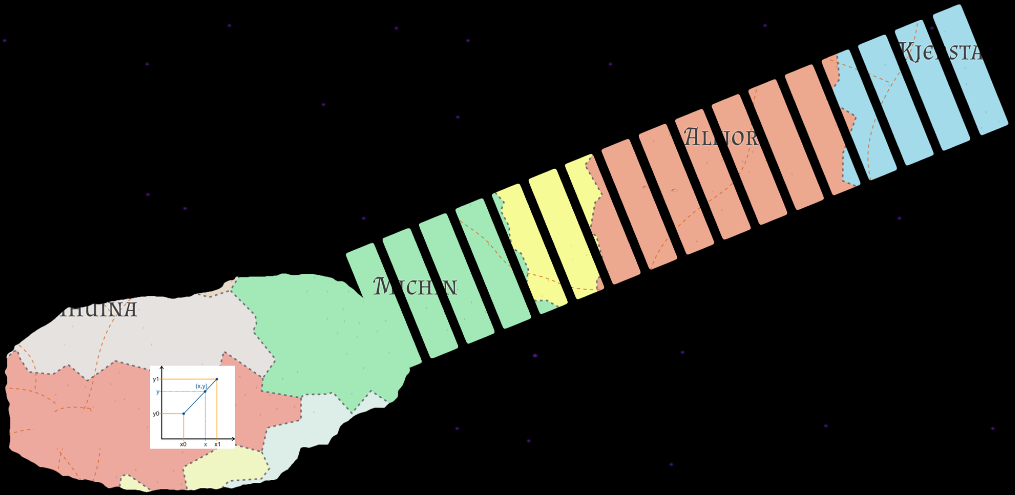
TECHNIQUE #3

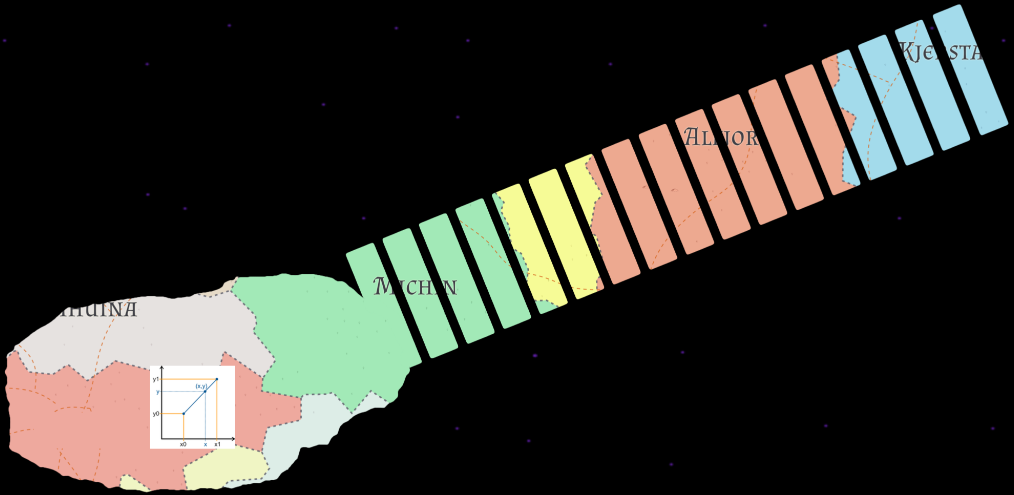


Analyse call stacks.

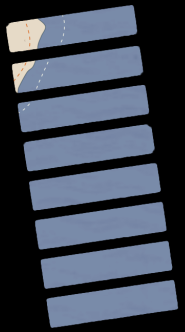

HUINA


y1
y
y0
(x,y)
x0
x
x1


HUINA
MICHAN
ALLIOR
KJESTA
y1
y
y0
(x,y)
x0
x
x1



HUINA
MICH N
ALLIOR
KJE STA



What is an interesting stack?

- A deep stack
- A common use case of the application

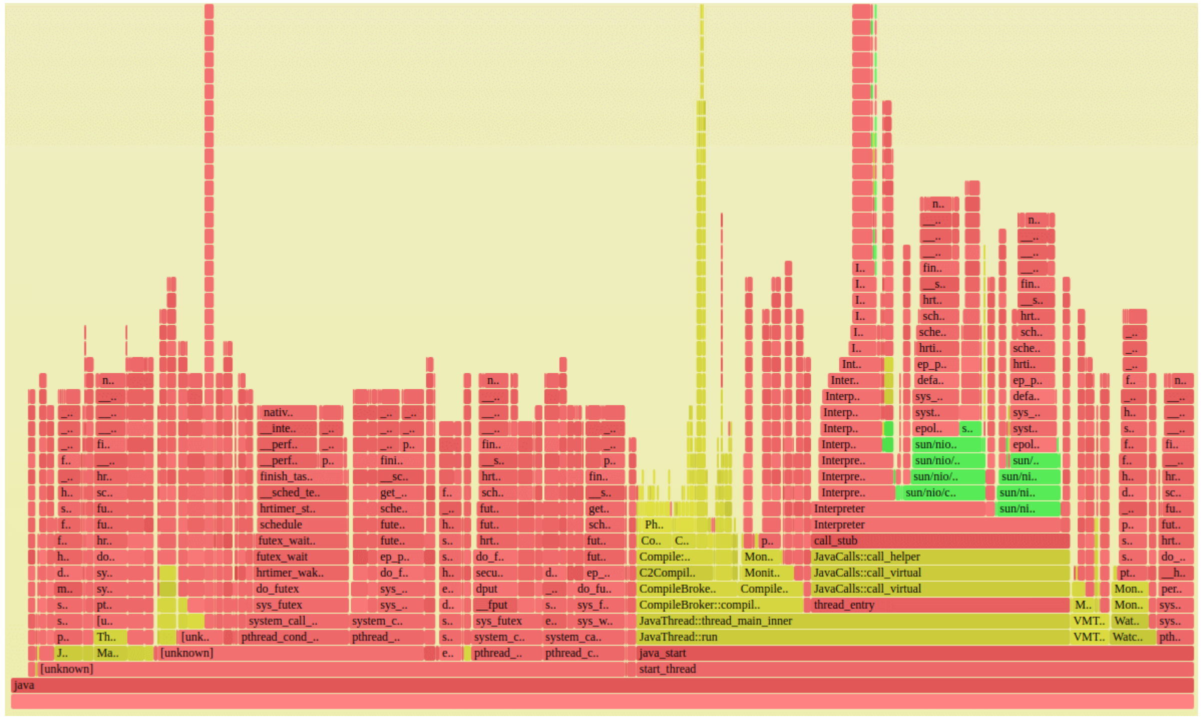
What is an interesting stack?

- A deep stack
- A common use case of the application

How to find an interesting stack?

- Your dev lead
- Business people for the common use case
- Your knowledge of the I/O to locate it

Flamegraphs



Exploring the code (3 techniques)

TECHNIQUE #1: Know your I/O frameworks.

TECHNIQUE #2: Find a stronghold.

TECHNIQUE #3: Analyse call stacks.

10 TECHNIQUES TO UNDERSTAND EXISTING CODE

OUTLINE:

Exploring the code (3 techniques)

Becoming a code speed-reader (4 techniques)

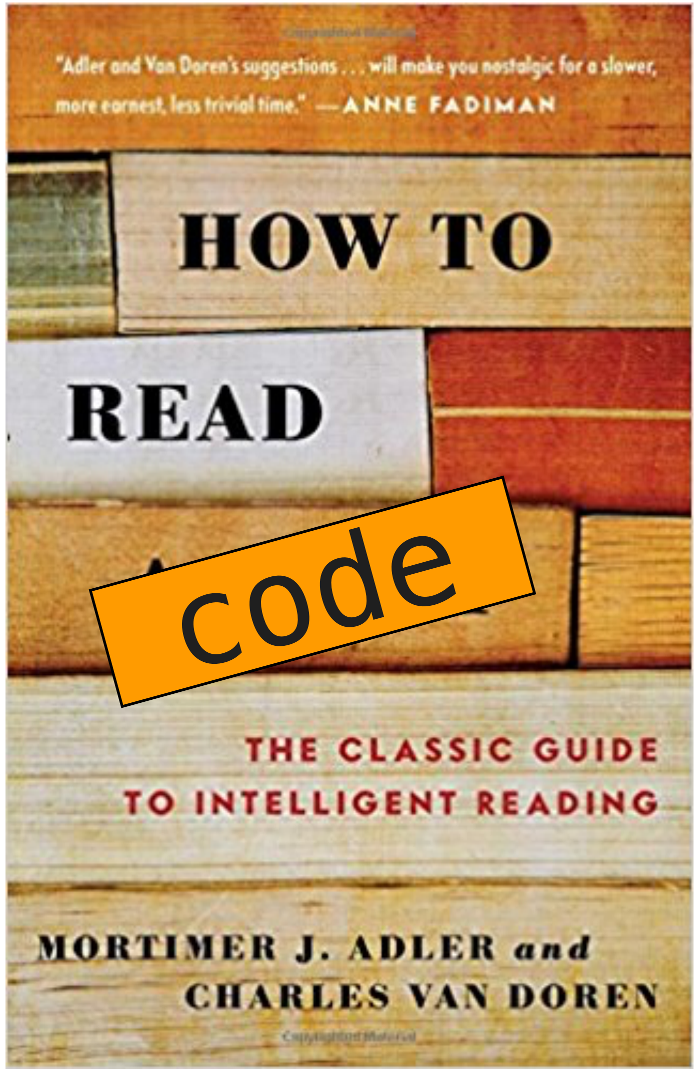
Understanding code in details (3 techniques)

Becoming a code speed-reader

(4 techniques)

doSomething()

A 2000-line function that will make your mind bend


"Adler and Van Doren's suggestions . . . will make you nostalgic for a slower, more earnest, less trivial time." —ANNE FADIMAN
HOW TO
READ
code
THE CLASSIC GUIDE TO INTELLIGENT READING
MORTIMER J. ADLER and CHARLES VAN DOREN

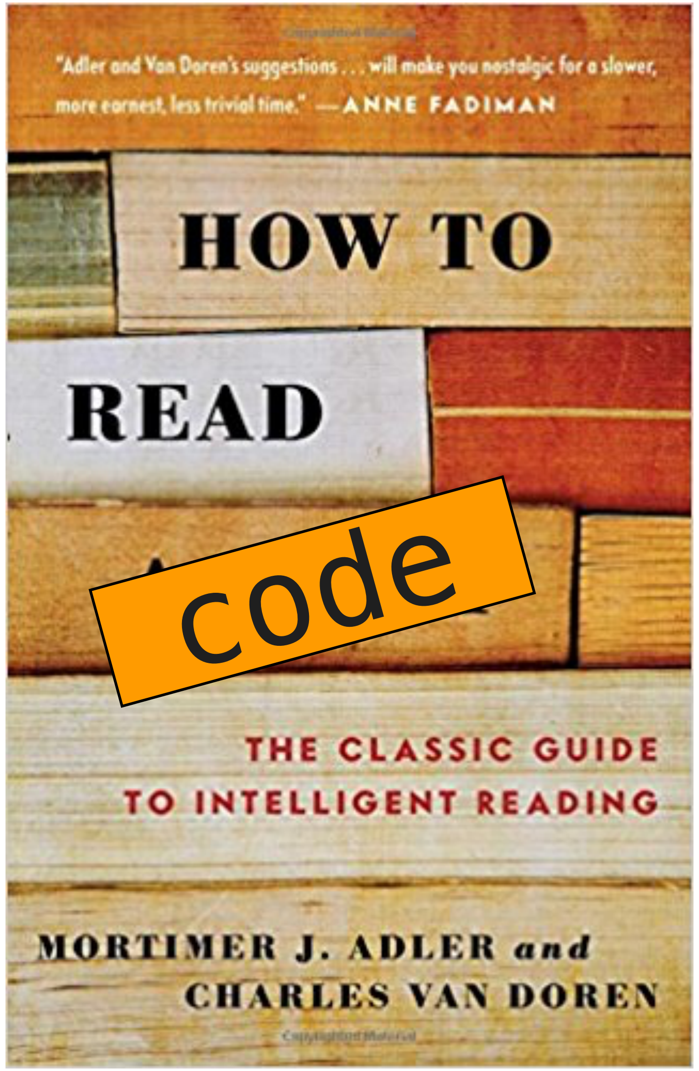
Don't read
cover to cover

Goal is not to
understand *everything*

Focus on where the
information is

TECHNIQUE #4

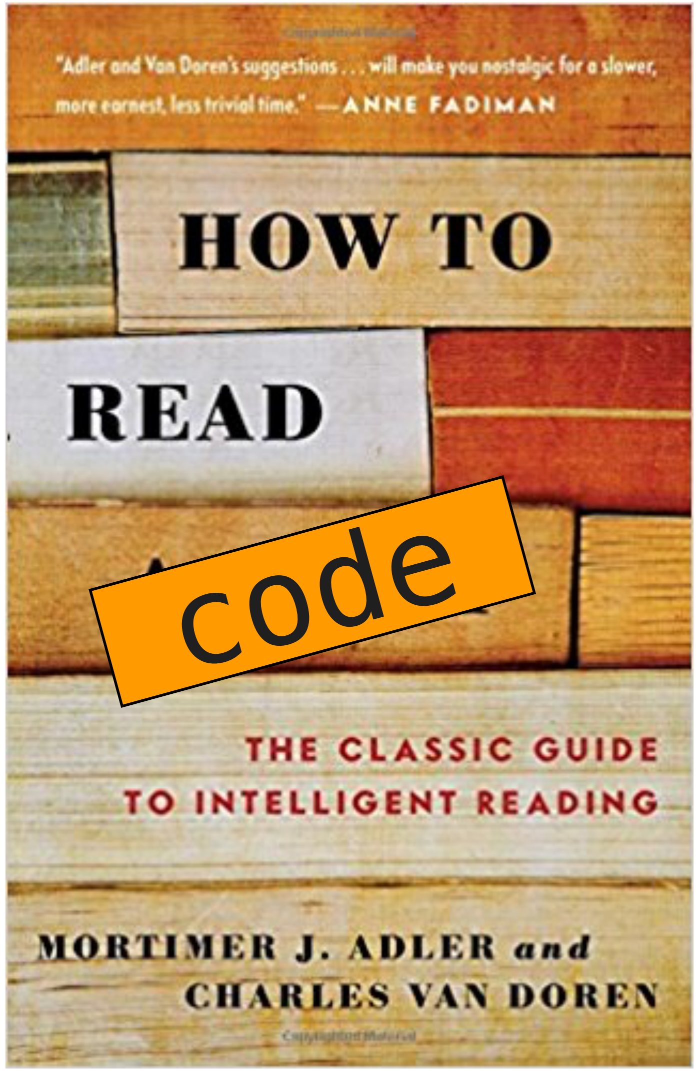
Start reading
from the end.


"Adler and Van Doren's suggestions . . . will make you nostalgic for a slower, more earnest, less trivial time." —ANNE FADIMAN
HOW TO
READ
code
THE CLASSIC GUIDE TO INTELLIGENT READING
MORTIMER J. ADLER and CHARLES VAN DOREN

TECHNIQUE #4

Information is
at the beginning
and the end

beginning = prototype
end = ?


"Adler and Van Doren's suggestions . . . will make you nostalgic for a slower, more earnest, less trivial time." —ANNE FADIMAN
HOW TO
READ
code
THE CLASSIC GUIDE TO INTELLIGENT READING
MORTIMER J. ADLER and CHARLES VAN DOREN

```
std::vector<MultipleAlignmentBlock> AlignmentGroup::loadAlignments(std::string const& chr, int start, int end) {

    IntervalTree* ivTree = index.getIntervalTree(chr);
    if (ivTree == NULL) return {};

    std::vector<Interval> intervals = ivTree.findOverlapping(start, end);
    if (intervals.empty()) {
        return {};
    }

    // Find the starting (left most) interval.  Alignment blocks do not overlap, so we can start at the
    // minimum file offset and just proceed until the end of the interval.
    Int startPosition = 0;
    for (Interval const& iv : intervals) {
        startPosition = Math.min(startPosition, iv.getValue());
    }

    IGVSeekableStream is;

    is = IGVSeekableStreamFactory.getInstance().getStreamFor(path);
    is.seek(startPosition);

    IGVBufferedReader reader(IGVInputStreamReader(is), 256000);

    std::vector<MultipleAlignmentBlock> alignments;

    std::string line;
    while (getLine(reader, line)) {
        if (startsWith(line, "a ")) {
            // TODO -- parse score (optional)
            MultipleAlignmentBlock block = parseBlock(reader);
            if(block.getEnd() < start) {
                continue;
            }
            if (block.getStart() > end || !block.getChr() == chr)) {
                break;
            } else {
                alignments.push_back(block);
            }
        }
    }
    return alignments;
}
```

@JoBoccara

TECHNIQUE #4

Start reading
from the end.

outputs.

- Returned value
- Out parameter
- Data member
- IO
- Global variable
- OUTPUT VIA EXCEPTION

Heuristics: look for outputs towards the end

TECHNIQUE #5

Find the frequent words.

```
bool CSetting::ReadValue( CRegKey &regKey, const wchar_t *valName )
{
    // bool, int, hotkey, color
    if (type==CSetting::TYPE_BOOL || (type==CSetting::TYPE_INT && this[1].type!=CSetting::TYPE_RADIO) || type==CSetting::TYPE_HOTKEY || type==CSetting::TYPE_HOTKEY_ANY || type==CSetting::TYPE_COLOR)
    {
        DWORD val;
        if (regKey.QueryDWORDValue(valName,val)==ERROR_SUCCESS)
        {
            if (type==CSetting::TYPE_BOOL)
                value=CComVariant(val?1:0);
            else
                value=CComVariant((int)val);
            return true;
        }
        return false;
    }
   // radio
    if (type==CSetting::TYPE_INT && this[1].type==CSetting::TYPE_RADIO)
    {
        ULONG len;
        DWORD val;
        if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            CString text;
            regKey.QueryStringValue(valName,text.GetBuffer(len),&len);
            text.ReleaseBuffer(len);
            val=0;
            for (const CSetting *pRadio=this+1;pRadio->type==CSetting::TYPE_RADIO;pRadio++,val++)
            {
                if (_wcsicmp(text,pRadio->name)==0)
                {
                    value=CComVariant((int)val);
                    return true;
                }
            }
        }
        else if (regKey.QueryDWORDValue(valName,val)==ERROR_SUCCESS)
        {
            value=CComVariant((int)val);
            return true;
        }
        return false;
    }
   // string
    if (type>=CSetting::TYPE_STRING && type<CSetting::TYPE_MULTISTRING)
    {
        ULONG len;
        if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            value.vt=VT_BSTR;
            value.bstrVal=SysAllocStringLen(NULL,len-1);
            regKey.QueryStringValue(valName,value.bstrVal,&len);
            return true;
        }
        return false;
    }
   // multistring
    if (type==CSetting::TYPE_MULTISTRING)
    {
        ULONG len;
        if (regKey.QueryMultiStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            value.vt=VT_BSTR;
            value.bstrVal=SysAllocStringLen(NULL,len-1);
            regKey.QueryMultiStringValue(valName,value.bstrVal,&len);
            for (int i=0;i<(int)len-1;i++)
                if (value.bstrVal[i]==0)
                    value.bstrVal[i]='\n';
            return true;
        }
        else if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            value.vt=VT_BSTR;
            value.bstrVal=SysAllocStringLen(NULL,len);
            regKey.QueryStringValue(valName,value.bstrVal,&len);
            if (len>0)
            {
                value.bstrVal[len-1]='\n';
                value.bstrVal[len]=0;
            }
            return true;
        }
        return false;
    }
    Assert(0);
    return false;
}
```

Word	# occurrences
len	20
value	17
CSetting	15
if	14
type	13
regKey	11
valName	11
return	11
val	10
bstrVal	10
1	8
0	8
NULL	7
true	6
QueryStringValue	6
ERROR_SUCCESS	6
int	6
…	

@JoBoccara

```
bool CSetting::ReadValue( CRegKey &regKey, const wchar_t *valName )
{
    // bool, int, hotkey, color
    if (type==CSetting::TYPE_BOOL || (type==CSetting::TYPE_INT && this[1].type!=CSetting::TYPE_RADIO) || type==CSetting::TYPE_HOTKEY || type==CSetting::TYPE_HOTKEY_ANY || type==CSetting::TYPE_COLOR)
    {
        DWORD val;
        if (regKey.QueryDWORDValue(valName,val)==ERROR_SUCCESS)
        {
            if (type==CSetting::TYPE_BOOL)
                value=CComVariant(val?1:0);
            else
                value=CComVariant((int)val);
            return true;
        }
        return false;
    }
   // radio
    if (type==CSetting::TYPE_INT && this[1].type==CSetting::TYPE_RADIO)
    {
        ULONG len;
        DWORD val;
        if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            CString text;
            regKey.QueryStringValue(valName,text.GetBuffer(len),&len);
            text.ReleaseBuffer(len);
            val=0;
            for (const CSetting *pRadio=this+1;pRadio->type==CSetting::TYPE_RADIO;pRadio++,val++)
            {
                if (_wcsicmp(text,pRadio->name)==0)
                {
                    value=CComVariant((int)val);
                    return true;
                }
            }
        }
        else if (regKey.QueryDWORDValue(valName,val)==ERROR_SUCCESS)
        {
            value=CComVariant((int)val);
            return true;
        }
        return false;
    }
   // string
    if (type>=CSetting::TYPE_STRING && type<CSetting::TYPE_MULTISTRING)
    {
        ULONG len;
        if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            value.vt=VT_BSTR;
            value.bstrVal=SysAllocStringLen(NULL,len-1);
            regKey.QueryStringValue(valName,value.bstrVal,&len);
            return true;
        }
        return false;
    }
   // multistring
    if (type==CSetting::TYPE_MULTISTRING)
    {
        ULONG len;
        if (regKey.QueryMultiStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            value.vt=VT_BSTR;
            value.bstrVal=SysAllocStringLen(NULL,len-1);
            regKey.QueryMultiStringValue(valName,value.bstrVal,&len);
            for (int i=0;i<(int)len-1;i++)
                if (value.bstrVal[i]==0)
                    value.bstrVal[i]='\n';
            return true;
        }
        else if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            value.vt=VT_BSTR;
            value.bstrVal=SysAllocStringLen(NULL,len);
            regKey.QueryStringValue(valName,value.bstrVal,&len);
            if (len>0)
            {
                value.bstrVal[len-1]='\n';
                value.bstrVal[len]=0;
            }
            return true;
        }
        return false;
    }
    Assert(0);
    return false;
}
```

	Word	# occurrences
	len	20
	value	17
	CSetting	15
	if	14
	type	13
inputs	regKey	11
	valName	11
	return	11
	val	10
	bstrVal	10
	1	8
	0	8
	NULL	7
	true	6
	QueryStringValue	6
	ERROR_SUCCESS	6
	int	6
	…	

@JoBoccara

```
IatHookData *SetIatHook( IMAGE_DOS_HEADER *dosHeader, DWORD iatOffset, DWORD intOfset, const char *targetProc, void *newProc)
{
    IMAGE_THUNK_DATA *thunk=(IMAGE_THUNK_DATA*)PtrFromRva(dosHeader,iatOffset);
    IMAGE_THUNK_DATA *origThunk=(IMAGE_THUNK_DATA*)PtrFromRva(dosHeader,intOfset);
    for (;origThunk->u1.Function;origThunk++,thunk++)
    {
        if (origThunk->u1.Ordinal&IMAGE_ORDINAL_FLAG)
        {
            if (IS_INTRESOURCE(targetProc) && IMAGE_ORDINAL(origThunk->u1.Ordinal)==(uintptr_t)targetProc)
                break;
        }
        else
        {
            IMAGE_IMPORT_BY_NAME *import=(IMAGE_IMPORT_BY_NAME*)PtrFromRva(dosHeader,origThunk->u1.AddressOfData);
            if (!IS_INTRESOURCE(targetProc) && strcmp(targetProc,(char*)import->Name)==0)
                break;
        }
    }
    if (origThunk->u1.Function)
    {
        IatHookData *hook=g_IatHooks+g_IatHookCount;
        g_IatHookCount++;
        hook->jump[0]=hook->jump[1]=0x90; // NOP
        hook->jump[2]=0xFF; hook->jump[3]=0x25; // JUMP
#ifdef _WIN64
        hook->jumpOffs=0;
#else
        hook->jumpOffs=(DWORD)(hook)+8;
#endif
        hook->newProc=newProc;
        hook->oldProc=(void*)thunk->u1.Function;
        hook->thunk=thunk;
        DWORD oldProtect;
        VirtualProtect(&thunk->u1.Function,sizeof(void*),PAGE_READWRITE,&oldProtect);
        thunk->u1.Function=(DWORD_PTR)hook;
        VirtualProtect(&thunk->u1.Function,sizeof(void*),oldProtect,&oldProtect);
        return hook;
    }
    return NULL;
}
```

```
int CSettingsParser::ParseTreeRec( const wchar_t *str, std::vector<TreeItem> &items, CString *names, int level )
{
    size_t start=items.size();
    while (*str)
    {
        wchar_t token[256];
        str=GetToken(str,token,_countof(token),L", \t");
        if (token[0])
        {
            //
            bool bFound=false;
            for (int i=0;i<level;i++)
                if (_wcsicmp(token,names[i])==0)
                {
                    bFound=true;
                    break;
                }
                if (!bFound)
                {
                    TreeItem item={token,-1};
                    items.push_back(item);
                }
        }
    }
    size_t end=items.size();
    if (start==end) return -1;

    TreeItem item={L"",-1};
    items.push_back(item);

    if (level<MAX_TREE_LEVEL-1)
    {
        for (size_t i=start;i<end;i++)
        {
            wchar_t buf[266];
            Sprintf(buf,_countof(buf),L"%s.Items",items[i].name);
            const wchar_t *str2=FindSetting(buf);
            if (str2)
            {
                names[level]=items[i].name;
                // these two statements must be on separate lines. otherwise items[i] is evaluated before ParseTreeRec, but
                // the items vector can be reallocated inside ParseTreeRec, causing the address to be invalidated -> crash!
                int idx=ParseTreeRec(str2,items,names,level+1);
                items[i].children=idx;
            }
        }
    }
    return (int)start;
}
```

Word	#	span	proportion
items	11	44	85 %
i	11	33	63 %
token	6	15	29 %
if	6	31	60 %
1	5	24	46 %
level	5	43	83 %
int	5	48	92 %
names	4	43	83 %
str	4	7	13 %
wchar_t	4	37	71 %
ParseTreeRec	4	43	83 %
start	4	46	88 %
item	4	10	19 %
buf	4	3	6 %

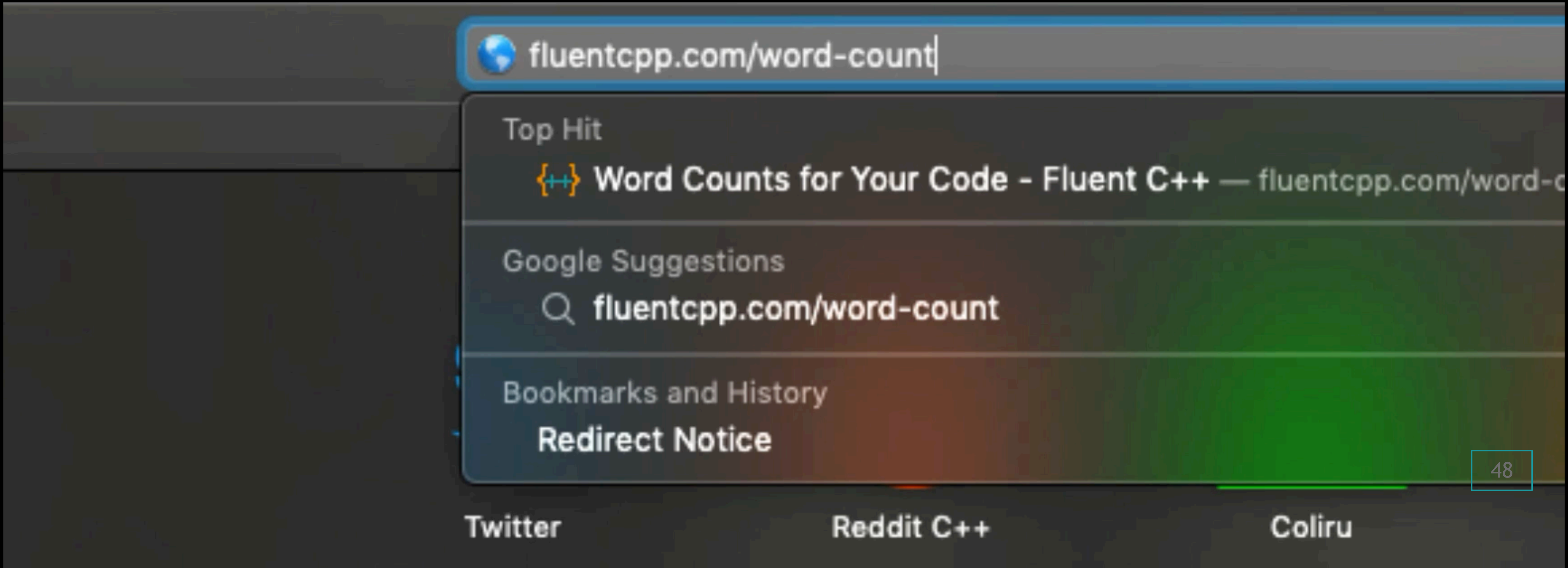
@JoBoccara

Tools:

- Your eyes
- IDE search highlighting
- Online tool

Tools:

- Your eyes
- IDE search highlighting
- Online tool



VARIOUS WAYS TO COUNT WORDS

- entire words
- wordsInCamelCase
- Case insensitive

COUNTING THE WORDS OF A MODULE

- Whole files
- Questions for newcomers

TECHNIQUE #6

Filter on control flow.

`else`

`if`

`switch`

`for` `while`

`case`

`do`

`try`

`catch`

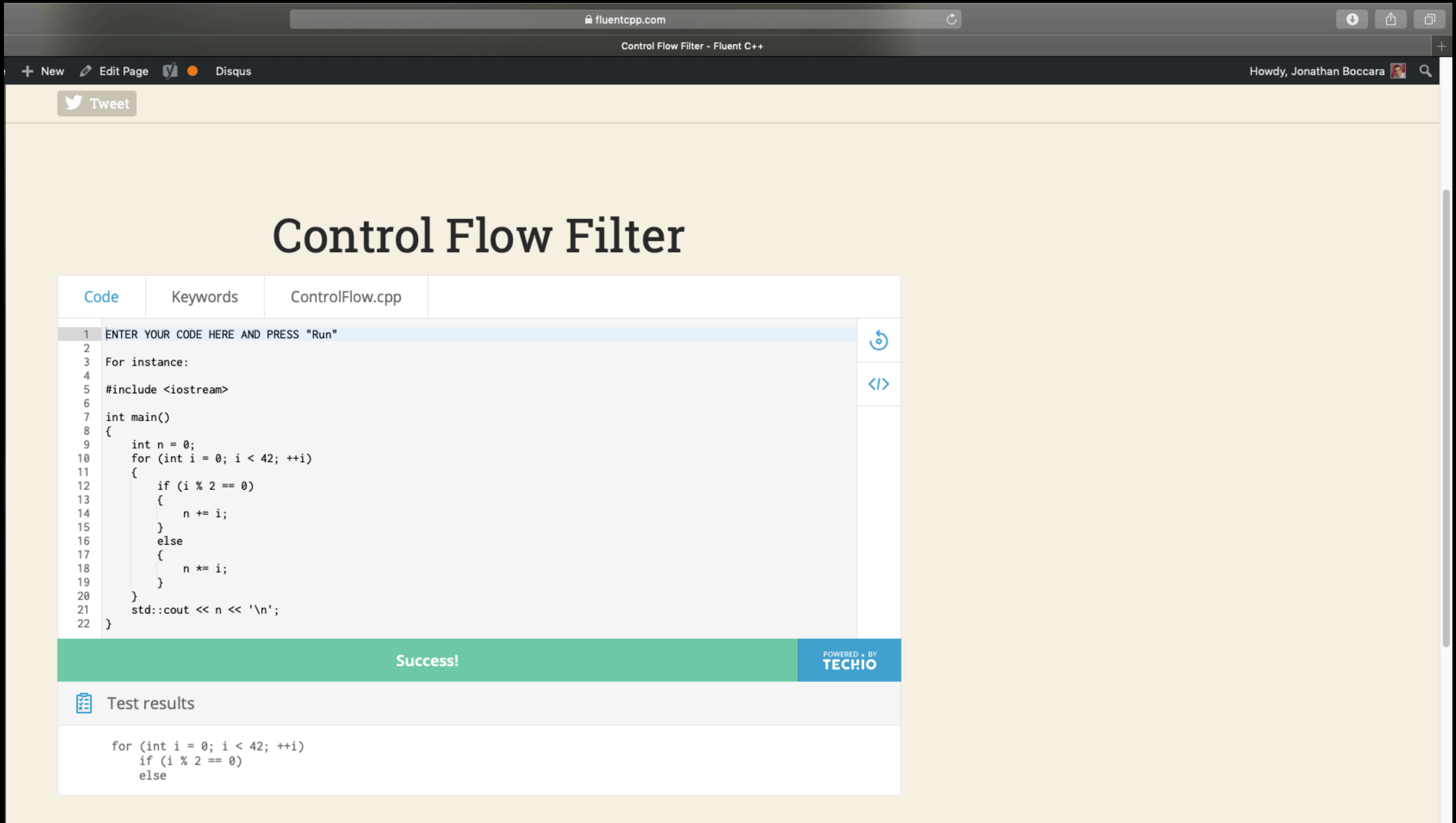
```
bool CSetting::ReadValue( CRegKey &regKey, const wchar_t *valName )
{
    // bool, int, hotkey, color
    if (type==CSetting::TYPE_BOOL || (type==CSetting::TYPE_INT && this[1].type!=CSetting::TYPE_RADIO) || type==CSetting::TYPE_HOTKEY || type==CSetting::TYPE_HOTKEY_ANY || type==CSetting::TYPE_COLOR)
    {
        DWORD val;
        if (regKey.QueryDWORDValue(valName,val)==ERROR_SUCCESS)
        {
            if (type==CSetting::TYPE_BOOL)
                value=CComVariant(val?1:0);
            else
                value=CComVariant((int)val);
            return true;
        }
        return false;
    }
   // radio
    if (type==CSetting::TYPE_INT && this[1].type==CSetting::TYPE_RADIO)
    {
        ULONG len;
        DWORD val;
        if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            CString text;
            regKey.QueryStringValue(valName,text.GetBuffer(len),&len);
            text.ReleaseBuffer(len);
            val=0;
            for (const CSetting *pRadio=this+1;pRadio->type==CSetting::TYPE_RADIO;pRadio++,val++)
            {
                if (_wcsicmp(text,pRadio->name)==0)
                {
                    value=CComVariant((int)val);
                    return true;
                }
            }
        }
        else if (regKey.QueryDWORDValue(valName,val)==ERROR_SUCCESS)
        {
            value=CComVariant((int)val);
            return true;
        }
        return false;
    }
   // string
    if (type>=CSetting::TYPE_STRING && type<CSetting::TYPE_MULTISTRING)
    {
        ULONG len;
        if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            value.vt=VT_BSTR;
            value.bstrVal=SysAllocStringLen(NULL,len-1);
            regKey.QueryStringValue(valName,value.bstrVal,&len);
            return true;
        }
        return false;
    }
   // multistring
    if (type==CSetting::TYPE_MULTISTRING)
    {
        ULONG len;
        if (regKey.QueryMultiStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            value.vt=VT_BSTR;
            value.bstrVal=SysAllocStringLen(NULL,len-1);
            regKey.QueryMultiStringValue(valName,value.bstrVal,&len);
            for (int i=0;i<(int)len-1;i++)
                if (value.bstrVal[i]==0)
                    value.bstrVal[i]='\n';
            return true;
        }
        else if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        {
            value.vt=VT_BSTR;
            value.bstrVal=SysAllocStringLen(NULL,len);
            regKey.QueryStringValue(valName,value.bstrVal,&len);
            if (len>0)
            {
                value.bstrVal[len-1]='\n';
                value.bstrVal[len]=0;
            }
            return true;
        }
        return false;
    }
    Assert(0);
    return false;
}
```

```
:g!/\(\<if\>\|\<else\>\|\<for\>\|\<while\>\|\<do\>\
|\<switch\>\|\<case\>\|\<try\>\|\<catch\>\)/d
```

```
if (type==CSetting::TYPE_BOOL || (type==CSetting::TYPE_INT &&
    this[1].type!=CSetting::TYPE_RADIO) || type==CSetting::TYPE_HOTKEY ||
    type==CSetting::TYPE_HOTKEY_ANY || type==CSetting::TYPE_COLOR)
    if (regKey.QueryDWORDValue(valName,val)==ERROR_SUCCESS)
        if (type==CSetting::TYPE_BOOL)
        else
if (type==CSetting::TYPE_INT && this[1].type==CSetting::TYPE_RADIO)
    if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        for (const CSetting *pRadio=this+1;pRadio->type==CSetting::TYPE_RADIO;pRadio++,val++)
            if (_wcsicmp(text,pRadio->name)==0)
    else if (regKey.QueryDWORDValue(valName,val)==ERROR_SUCCESS)
if (type>=CSetting::TYPE_STRING && type<CSetting::TYPE_MULTISTRING)
    if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
if (type==CSetting::TYPE_MULTISTRING)
    if (regKey.QueryMultiStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        for (int i=0;i<(int)len-1;i++)
            if (value.bstrVal[i]==0)
    else if (regKey.QueryStringValue(valName,NULL,&len)==ERROR_SUCCESS)
        if (len>0)
```

CSetting	14
if	14
type	13
len	6
ERROR_SUCCESS	6
regKey	6
valName	6
1	4

fluentcpp.com/control-flow-filter



Control Flow Filter

```
ENTER YOUR CODE HERE AND PRESS "Run"

For instance:

#include <iostream>

int main()
{
    int n = 0;
    for (int i = 0; i < 42; ++i)
    {
        if (i % 2 == 0)
        {
            n += i;
        }
        else
        {
            n *= i;
        }
    }
    std::cout << n << '\n';
}
```

```
for (int i = 0; i < 42; ++i)
    if (i % 2 == 0)
    else
```

TECHNIQUE #7

Scan for the main action.

THE

80 20

RULE

THE

90 10

RULE

QUICK scan for the main action

- Speed over accuracy
- A second pass is possible

Things that are not the main action:

- Secondary variables
- Special cases
- Complicated stuff (in all likelyhood)
- …

```
void AlignementGroup::processAlignments(std::string const& chr, std::vector<Alignment> const& alignments) {

    Genome* genome = GenomeManager.getInstance().getCurrentGenome();
    chr = genome == NULL ? chr : genome.getCanonicalChrName(chr);

    auto insertionMapIt = insertionMaps.find(chr);
    if(insertionMapIt == end(insertionMaps)) {
       insertionMaps[chr] = std::map<int, InsertionMarker>{};
    }
    std::map<int, InsertionMarker>& insertionMap = *insertionMaps.find(chr);
    auto positionsIt = positionsMap.find(chr);
    if(positionsIt == end(positionsMap)) {
       positionsMap[chr] = std::vector<int>{});
    }
    std::vector<int>& positions = *positionsMap.find(chr);
    int minLength = 0;
    if (PreferencesManager.getPreferences().getAsBoolean(SAM_HIDE_SMALL_INDEL)) {
        minLength = PreferencesManager.getPreferences().getAsInt(SAM_SMALL_INDEL_BP_THRESHOLD);
    }
   for (Alignment const& a : alignments) {
        std::vector<AlignmentBlock> const& blocks = a.getInsertions();
        if (!blocks.empty()) {
            for (AlignmentBlock const& block : blocks) {

                if (block.getBases().length < minLength) continue;

                int key = block.getStart();
                InsertionMarker insertionMarkerIt = insertionMap.find(key);
                if (insertionMarkerIt == insertionMap.end()) {
                    InsertionMarker insertionMarker(block.getStart(), block.getLength());
                    insertionMap[key] = insertionMarker;
                    positions.push_back(block.getStart());
                } else {
                    insertionMarker.size = std::max(insertionMarker.size, block.getLength());
                }
            }
        }
    }
    std::transform(begin(insertionMap), end(insertionMap), std::back_inserter(positions), getFirst);
    std::sort(begin(positions), end(positions));
}
```

@JoBoccara

Word	#	span	proportion
chr	10	15	36 %
std	10	40	95 %
block	7	12	29 %
int	6	20	48 %
insertionMap	6	30	71 %
if	6	23	55 %
positions	5	26	61 %
end	5	34	81 %
const	5	23	55 %
find	5	23	55 %
…	…	…	…

```
void AlignementGroup::processAlignments(std::string const& chr, std::vector<Alignment> const& align

    Genome* genome = GenomeManager.getInstance().getCurrentGenome();
    chr = genome == NULL ? chr : genome.getCanonicalChrName(chr);

    auto insertionMapIt = insertionMaps.find(chr);
    if(insertionMapIt == end(insertionMaps)) {
       insertionMaps[chr] = std::map<int, InsertionMarker>{};
    }
    std::map<int, InsertionMarker>& insertionMap = *insertionMaps.find(chr);
    auto positionsIt = positionsMap.find(chr);
    if(positionsIt == end(positionsMap)) {
       positionsMap[chr] = std::vector<int>{});
    }
    std::vector<int>& positions = *positionsMap.find(chr);
    int minLength = 0;
    if (PreferencesManager.getPreferences().getAsBoolean(SAM_HIDE_SMALL_INDEL)) {
        minLength = PreferencesManager.getPreferences().getAsInt(SAM_SMALL_INDEL_BP_THRESHOLD);
    }
   for (Alignment const& a : alignments) {
        std::vector<AlignmentBlock> const& blocks = a.getInsertions();
        if (!blocks.empty()) {
            for (AlignmentBlock const& block : blocks) {

                if (block.getBases().length < minLength) continue;

                int key = block.getStart();
                InsertionMarker insertionMarkerIt = insertionMap.find(key);
                if (insertionMarkerIt == insertionMap.end()) {
                    InsertionMarker insertionMarker(block.getStart(), block.getLength());
                    insertionMap[key] = insertionMarker;
                    positions.push_back(block.getStart());
                } else {
                    insertionMarker.size = std::max(insertionMarker.size, block.getLength());
                }
            }
        }
    }
    std::transform(begin(insertionMap), end(insertionMap), std::back_inserter(positions), getFirst)
    std::sort(begin(positions), end(positions));
}
```

@JoBoccara

```
void AlignementGroup::processAlignments(std::string const& chr, std::vector<Alignment> const& alignments) {

    Genome* genome = GenomeManager.getInstance().getCurrentGenome();
    chr = genome == NULL ? chr : genome.getCanonicalChrName(chr);

    auto insertionMapIt = insertionMaps.find(chr);
    if(insertionMapIt == end(insertionMaps)) {
       insertionMaps[chr] = std::map<int, InsertionMarker>{};
    }
    std::map<int, InsertionMarker>& insertionMap = *insertionMaps.find(chr);
    auto positionsIt = positionsMap.find(chr);
    if(positionsIt == end(positionsMap)) {
       positionsMap[chr] = std::vector<int>{});
    }
    std::vector<int>& positions = *positionsMap.find(chr);
    int minLength = 0;
    if (PreferencesManager.getPreferences().getAsBoolean(SAM_HIDE_SMALL_INDEL)) {
        minLength = PreferencesManager.getPreferences().getAsInt(SAM_SMALL_INDEL_BP_THRESHOLD);
    }
   for (Alignment const& a : alignments) {
        std::vector<AlignmentBlock> const& blocks = a.getInsertions();
        if (!blocks.empty()) {
            for (AlignmentBlock const& block : blocks) {

                if (block.getBases().length < minLength) continue;

                int key = block.getStart();
                InsertionMarker insertionMarkerIt = insertionMap.find(key);
                if (insertionMarkerIt == insertionMap.end()) {
                    InsertionMarker insertionMarker(block.getStart(), block.getLength());
                    insertionMap[key] = insertionMarker;
                    positions.push_back(block.getStart());
                } else {
                    insertionMarker.size = std::max(insertionMarker.size, block.getLength());
                }
            }
        }
    }
    std::transform(begin(insertionMap), end(insertionMap), std::back_inserter(positions), getFirst);
    std::sort(begin(positions), end(positions));
}
```

```
if(insertionMapIt == end(insertionMaps)) {
   if(positionsIt == end(positionsMap)) {
   if (PreferencesManager.getPreferences().getAsBoolean(SAM_HIDE_SMALL_INDEL)) {
  for (Alignment const& a : alignments) {
       if (!blocks.empty()) {
           for (AlignmentBlock const& block : blocks) {
               if (block.getBases().length < minLength) continue;
               if (insertionMarkerIt == insertionMap.end()) {
               } else {
```

```
void AlignementGroup::processAlignments(std::string const& chr, std::vector<Alignment> const& alignments) {

        Genome* genome = GenomeManager.getInstance().getCurrentGenome();
        chr = genome == NULL ? chr : genome.getCanonicalChrName(chr);

        auto insertionMapIt = insertionMaps.find(chr);
        if(insertionMapIt == end(insertionMaps)) {
           insertionMaps[chr] = std::map<int, InsertionMarker>{};
        }
        std::map<int, InsertionMarker>& insertionMap = *insertionMaps.find(chr);
        auto positionsIt = positionsMap.find(chr);
        if(positionsIt == end(positionsMap)) {
           positionsMap[chr] = std::vector<int>{});
        }
        std::vector<int>& positions = *positionsMap.find(chr);
        int minLength = 0;
        if (PreferencesManager.getPreferences().getAsBoolean(SAM_HIDE_SMALL_INDEL)) {
            minLength = PreferencesManager.getPreferences().getAsInt(SAM_SMALL_INDEL_BP_THRESHOLD);
        }
       for (Alignment const& a : alignments) {
            std::vector<AlignmentBlock> const& blocks = a.getInsertions();
            if (!blocks.empty()) {
                for (AlignmentBlock const& block : blocks) {

                    if (block.getBases().length < minLength) continue;

                    int key = block.getStart();
                    InsertionMarker insertionMarkerIt = insertionMap.find(key);
                    if (insertionMarkerIt == insertionMap.end()) {
                        InsertionMarker insertionMarker(block.getStart(), block.getLength());
                        insertionMap[key] = insertionMarker;
                        positions.push_back(block.getStart());
                    } else {
                        insertionMarker.size = std::max(insertionMarker.size, block.getLength());
                    }
                }
            }
        }
        std::transform(begin(insertionMap), end(insertionMap), std::back_inserter(positions), getFirst);
        std::sort(begin(positions), end(positions));
}
```

@JoBoccara

Becoming a code speed-reader (4 techniques)

TECHNIQUE #4: Start reading from the outputs.

TECHNIQUE #5: Find the frequent words.

TECHNIQUE #6: Filter on control flow.

TECHNIQUE #7: Scan for the main action.

10 TECHNIQUES TO UNDERSTAND EXISTING CODE

OUTLINE:

Exploring the code (3 techniques)

Becoming a code speed-reader (4 techniques)

Understanding code in details (3 techniques)

Understanding code in detail

(3 techniques)

TECHNIQUE #8

Decouple the code.

```
class Order
{
public:
    Order(double price, Country const& originCountry, Country const& destinationCountry);

    double getTotalPrice() const;
    // ...
    void process();
    // ...

private:
    void processTaxes();

    // ...
    double price_;
    double totalPrice_;
    // ...
    Country originCountry_;
    Country destinationCountry_;
    // ...
};
```

```
void Order::process()
{
    // ...
    processTaxes();
    // ...
}
```

```
void Order::processTaxes()
{
   double taxValue = price_ * getTaxRate(destinationCountry_);
   double internationalTaxCredit = getInternatinalTaxCreditValue(price_, originCountry_,
                                                                  destinationCountry_);
   double taxCut = getTaxCutRate(price_) * price_;
   totalPrice_ = price_ + taxValue;
   double taxReduction = std::min(internationalTaxCredit + taxCut, totalPrice_);
   totalPrice_ -= taxReduction;
}
```

```
class Order
{
public:
    // ...

private:
    void processTaxes();
    // ...
    double price_;
    double totalPrice_;
    // ...
    Country originCountry_;
    Country destinationCountry_;
    // ...
};
```

The compiler helps
locate data member accesses

```
void processTaxes(Order& order)
{
   double taxValue = order.getPrice() * getTaxRate(order.getDestinationCountry());
   double internationalTaxCredit = getInternatinalTaxCreditValue(order.getPrice(),
                                       order.getOriginCountry(), order.getDestinationCountry());
   double taxCut = getTaxCutRate(order.getPrice()) * order.getPrice();
   order.setTotalPrice(order.getPrice() + taxValue);
   double taxReduction = std::min(internationalTaxCredit + taxCut, order.getTotalPrice());
   order.setTotalPrice(order.getTotalPrice() - taxReduction);
}
```

```
class Order
{
public:
    // ...

private:
    // ...
    double price_;
    double totalPrice_;
    // ...
    Country originCountry_;
    Country destinationCountry_;
    // ...
};

void processTaxes(Order& order);
```

Getters
- getPrice()
- getOriginCountry()
- getDestinationCountry()
- getTotalPrice()

Setters
- setTotalPrice()

```
double processTaxes(double price, Country const& originCountry,
                                    Country const& destinationCountry, double totalPrice)
{
    double taxValue = price * getTaxRate(destinationCountry);
    double internationalTaxCredit = getInternatinalTaxCreditValue(price, originCountry,
                                                                          destinationCountry);
    double taxCut = getTaxCutRate(price) * price;
    totalPrice = price + taxValue;
    double taxReduction = std::min(internationalTaxCredit + taxCut, totalPrice);
    return totalPrice - taxReduction;
}
```

```
double priceAfterTaxes(double price, Country const& originCountry, Country const&
                                                                        destinationCountry)
{
    double taxValue = price * getTaxRate(destinationCountry);
    double internationalTaxCredit = getInternatinalTaxCreditValue(price, originCountry,
                                                                        destinationCountry);
    double taxCut = getTaxCutRate(price) * price;
    double totalPrice = price + taxValue;
    double taxReduction = std::min(internationalTaxCredit + taxCut, totalPrice);
    return totalPrice - taxReduction;
}
```

```
void Order::process()
{
    // ...
    totalPrice_ = priceAfterTaxes(price_, originCountry_, destinationCountry_);
}
```

```
double priceAfterTaxes(double price, Country const& originCountry, Country const&
                                                                    destinationCountry)
{
    double taxValue = price * getTaxRate(destinationCountry);
    double internationalTaxCredit = getInternatinalTaxCreditValue(price, originCountry,
                                                                    destinationCountry);
    double taxCut = getTaxCutRate(price) * price;
    double totalPrice = price + taxValue;
    double taxReduction = std::min(internationalTaxCredit + taxCut, totalPrice);
    return totalPrice - taxReduction;
}
```

```
void Order::process()
{
    // ...
    priceAfterTaxes_ = priceAfterTaxes(price_, originCountry_, destinationCountry_);
}
```

- Scratch refactoring: throw it away?
- Keep new naming

TECHNIQUE #9

Know the conventions

Prototypes

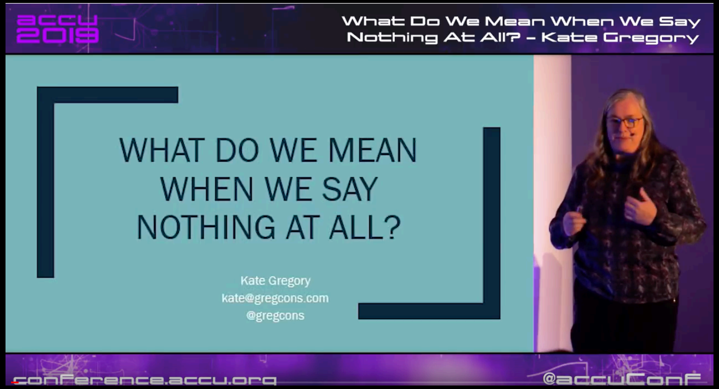
`Y f(X& value);`	Object is modified
`Y f(X&& value);`	Object is sunk
`Y f(X const& value);`	Object is input and you're dealing with an East const person
`Y f(X value);`	Object is copied or sunk
`Y f(std::optional<X> value);`	The function can work without the object
`std::optional<Y> f(X const& value);`	The function may or may not return a value
`Y f(X* value);`	Like & but can be null
`Y f(std::unique_ptr<X> value);`	The object is sunk
`Y f(std::shared_ptr<X> value);`	Elaborate memory handling at play
`Y f(std::unique_ptr<X> const& value);`	Be wary
`Y f(std::unique_ptr<X>* value);`	This will blow up one day

Standard types

`std::void_t`	Detection idiom
`std::set`	Sorted + unique
`std::unordered_map`	Lookup speed is what matters
`std::list`	Be wary

Are conventions always followed?

No. Look around to find out if they are in your code


accu 2019

What Do We Mean When We Say Nothing At All? – Kate Gregory

WHAT DO WE MEAN WHEN WE SAY NOTHING AT ALL?

Kate Gregory
kate@gregcons.com
@gregcons

conference.accu.org

@accuConf

But to benefit from them you have to know them

TECHNIQUE #10

Team up.

- A fresh look
- Higher returns
- Rubber-ducking
- Sum of knowledge
- Sustained focus

Understanding code in detail

TECHNIQUE #8: Decouple the code.

TECHNIQUE #9: Know the conventions.

TECHNIQUE #10: Team up.

10 TECHNIQUES TO UNDERSTAND EXISTING CODE

Exploring the code

- #1: Know your I/O frameworks.
- #2: Find a stronghold.
- #3: Analyse call stacks.

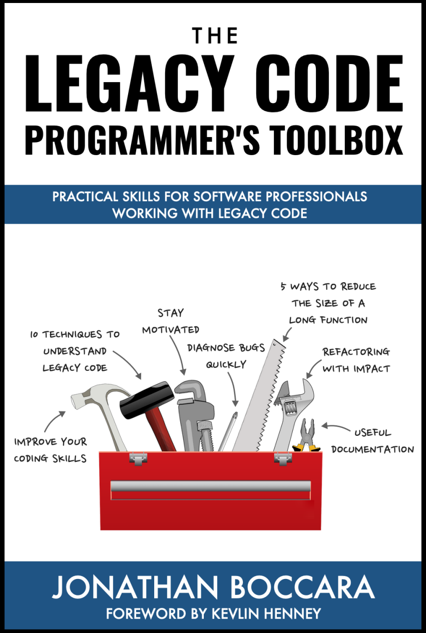
Becoming a code speed-reader

- #4: Start reading from the outputs.
- #5: Count words.
- #6: Filter on control flow.
- #7: Scan for the main action.

Understanding code in details

- #8: Decouple the code.
- #9: Know the conventions.
- #10: Team up.

ABOUT WORKING WITH EXISTING CODE



THE LEGACY CODE PROGRAMMER'S TOOLBOX

PRACTICAL SKILLS FOR SOFTWARE PROFESSIONALS WORKING WITH LEGACY CODE

10 TECHNIQUES TO UNDERSTAND LEGACY CODE
STAY MOTIVATED
DIAGNOSE BUGS QUICKLY
5 WAYS TO REDUCE THE SIZE OF A LONG FUNCTION
REFACTORING WITH IMPACT
IMPROVE YOUR CODING SKILLS
USEFUL DOCUMENTATION

JONATHAN BOCCARA

FOREWORD BY KEVLIN HENNEY

leanpub.com/legacycode

10 TECHNIQUES TO UNDERSTAND EXISTING CODE

Thank you!

10 TECHNIQUES TO UNDERSTAND EXISTING CODE

Exploring the code

- #1: Know your I/O frameworks.
- #2: Find a stronghold.
- #3: Analyse call stacks.

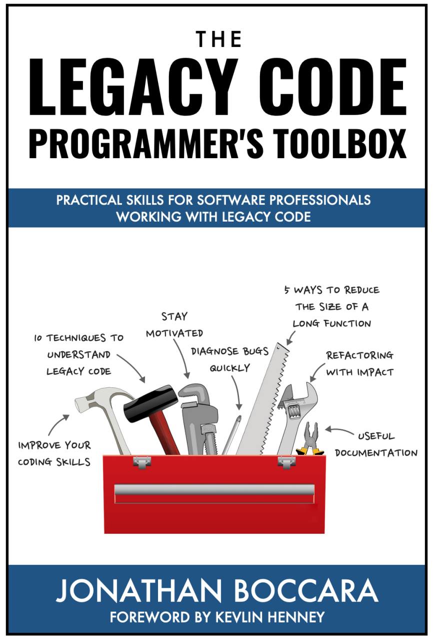
Becoming a code speed-reader

- #4: Start reading from the outputs.
- #5: Count words.
- #6: Filter on control flow.
- #7: Scan for the main action.

Understanding code in details

- #8: Decouple the code.
- #9: Know the conventions.
- #10: Team up.

ABOUT WORKING WITH EXISTING CODE



THE
LEGACY CODE
PROGRAMMER'S TOOLBOX

PRACTICAL SKILLS FOR SOFTWARE PROFESSIONALS WORKING WITH LEGACY CODE

10 TECHNIQUES TO UNDERSTAND LEGACY CODE
STAY MOTIVATED
DIAGNOSE BUGS QUICKLY
5 WAYS TO REDUCE THE SIZE OF A LONG FUNCTION
REFACTORING WITH IMPACT
IMPROVE YOUR CODING SKILLS
USEFUL DOCUMENTATION

JONATHAN BOCCARA
FOREWORD BY KEVLIN HENNEY

leanpub.com/legacycode

@JoBoccara