The Care & Feeding of C++'s Dragons

Chandler Carruth
GoingNative 2013

Dragons? C++?

C++ is hard

History: the legacy of C

History: the legacy of old C++

Subtlety:
“C++ has indeed become too ‘expert friendly’”

Complexity...

COMPLEXITY...
THE CAUSE OF, AND SOLUTION TO, ALL PROGRAMMING PROBLEMS
nerator.net

The cost of complexity is exponential.

The application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software.

ENGINEERING
IT'S LIKE MATH
BUT LOUDER

In Reality...

- Factoring the functionality
- Providing layering and boundaries
- Building abstractions
- Defining specifications

Simplicity ≫ Complexity

Build abstractions with simple interfaces around complex functionality

Remember:
Clever is not a Compliment!

But that takes time...
and energy...

You need tools to stay productive!
STOP
DZY 3671

Automate & Accelerate the manual tasks

LLVM

&

Clang

Historically, GCC didn’t cut it...

VOGUE NAILS
Specials
WHOLE BODY WAX
EYEBROW WAX $10
SPA PEDICURE
FREE
503-235-3999
WALK-INS WELCOME

But this isn’t fair at all...

GCC is improving faster than ever,
Clang has improved directly as a result

Still, LLVM remains our preferred platform for C++ tools...

LLVM: An OSS project

LLVM: Compiler Infrastructure

Clang: Production C++11 Compiler

LLD: An LLVM Linker

LLDB: An LLVM Debugger

LLVM: A Complete C++ Toolchain

The first tool...

Compilers

Principles, Techniques, and Tools

Clang/LLVM is a production quality C++11 compiler...

...and is rapidly implementing the draft C++14!

Generates high-quality code,
and does so reasonably quickly

Diagnoses broken
code really well
(See last year's talk!)

Lots of important warnings...

But let’s talk about *tools*!

Authorship and *Modification*

Making code look the way you want it

or

Refactoring!

We set out to build Fowler's catalog of refactoring tools...

But none of them worked

```
void foo(...) {
  for (...) {
    if (...) {
      if (...)
{
  Widget magic_widget{...};
  magic_widget.build(really, long, argument, list, barely, fits);
        bar(magic_widget,
                    some_other_param,
                    another_param);
}
       else
          bar(...);
```

Refactoring
Reformatting
Time to change
Lines of code

Refactoring
OUT OF TIME!!!
Reformatting
Time to change
Lines of code

What is the most important tool to reduce wasted time?

Formatting Whitespace

Yea, this stuff...

Because your code is a unique and beautiful snowflake

moments later in code review...
WAIT HOW DARE YOU
FORMAT IT THAT WAY!

BECAUSE THAT IS THE
ONE TRUE FORMATTING!

Remember:
Our challenge is complexity

One way we deal with it is through patterns

Well formatted code forms a pattern that simplifies it

Unfortunately, indenting is easy,
but formatting C++ is *hard*!

Where do you break a line?

Enough slides, let’s have a demo!

YAY LIVE DEMO!!!

Another tool to save you time...

clang-modernize: let’s try it!

YAY MORE DEMO!!!

And now for something completely different...

Tools for debugging and protecting your code

DO NOT WANT!!!

We use both static and dynamic analysis tools

80% of bugs caught statically
Very low (non-zero) false positives

20% of bugs caught dynamically
Zero false positives!

3 components of static analysis

The first are compiler warnings
(we've been over these...)

An interesting aspect to these are *model checking* warnings

The second component is the offline static analyzer

<demo static analyzer>

The final component is offline inference tooling to automate annotating your code

(coming soon...)

Static analysis for C++ is still really new in Clang, lots left to do

But what about those *really bad* bugs we miss statically?

THESE BUGS, REMEMBER?

Dynamic analysis: Valgrind? PIN? Others?

They haven’t caught on...

... maybe because they're too slow
... and they miss too many bugs
... and they don't really help you fix

LLVM Sanitizers: Compiler instrumentation dynamic analysis

The number of sanitizers is three:

1) Address Sanitizer
2) Thread Sanitizer
3) Memory Sanitizer

These are based on *shadow memory*, can't be combined

The number of sanitizers is four~~three~~:

1) Address Sanitizer
2) Thread Sanitizer
3) Memory Sanitizer
4) Undefined Behavior Sanitizer

Don’t forget: dynamic analysis only works if you *test your code*!

Let’s walk through some bugs...

OMG MORE DEMO!!!

to sum up...

Complexity is the biggest challenge for C++ today

Tools can help

Automate the manual tasks
Accelerate the complex tasks

Find bugs faster -- 80% offline

Use dynamic analysis for the long tail of subtle, complex bugs

Combine it with thorough testing to protect your software

These tools help make writing correct & reliable C++ fun & easy

Now get back to writing code
and engineering software!

Thanks to all the LLVM hackers!
(there are too many to list!)

Oh, and one more thing...

http://llvm.org/builds/