The Most Important Design Guideline is Testability

JODY HAGINS

The Most Important Design Guideline is Testability

JODY HAGINS

MAYSTREET

CppCon 2024

The Most Important Design Guideline is Testability

Jody Hagins
jhagins@dev.null

MAYSTREET

CppCon 2024

The Most Important Design Guideline is Testability

Jody Hagins
jhagins@dev.null

MAYSTREET

CppCon 2024

The Most Important Design Guideline is Testability

Jody Hagins
jhagins@maystreet.com
coachhagins@gmail.com

Grad School and Winning the Lottery

Remember This?

NDC 2014

4

https://www.youtube.com/watch?v=a76Z__kQJdo&t=83s

The Most Important Design Guideline

NDC 2014

4

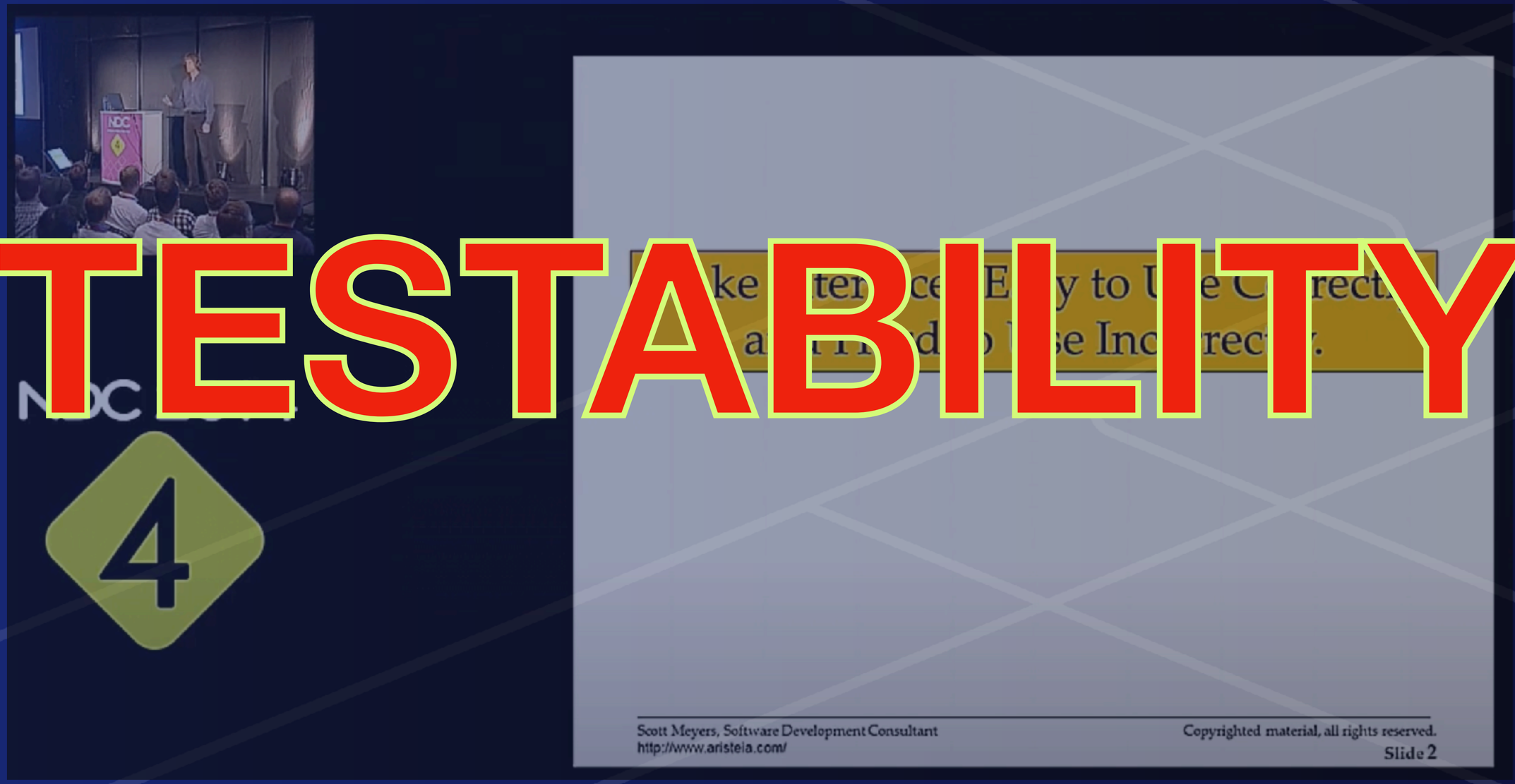
Make Interfaces Easy to Use Correctly and Hard to Use Incorrectly.

Scott Meyers, Software Development Consultant
http://www.aristeia.com/

Slide 2

https://www.youtube.com/watch?v=a76Z__kQJdo&t=83s

The Most Important API Design Guideline?



https://www.youtube.com/watch?v=a76Z__kQJdo&t=83s

Why is Testing Important?

Why is Testing Important?

- We are human!

Why is Testing Important?

- We are human!
  - Making Mistakes: Every human is really, really good at making mistakes

Why is Testing Important?

- We are human!
  - Making Mistakes: Every human is really, really good at making mistakes
  - Being annoyed when OTHER people make mistakes

Testability

Testability

- Unit Tests

Testability

- Unit Tests
- TDD

Testability

I often struggle with how to make my classes testable. I sometimes find that the issue is hidden state. I then have to find ways to expose and manipulate that state in test cases. Sometimes that causes new problems typically exposing interfaces that exist solely for unit testing. Putting them in a separate public section helps but is less than satisfying. Perhaps another possibly better approach would be to provide a mockable component. But that complicates the programmer's API with more template parameters. I could keep going along this line but I won't.

Testability

I often struggle with how to make my classes testable. I sometimes find that the issue is hidden state. I then have to find ways to expose and manipulate that state in test cases. Sometimes that causes new problems typically exposing interfaces that exist solely for unit testing. Putting them in a separate public section helps but is less than satisfying. Perhaps another possibly better approach would be to provide a mockable component. But that complicates the programmer's API with more template parameters. I could keep going along this line but I won't.

Testability

I often struggle with how to make my classes testable. I sometimes find that the issue is hidden state. I then have to find ways to expose and manipulate that state in test cases. Sometimes that causes new problems typically exposing interfaces that exist solely for unit testing. Putting them in a separate public section helps but is less than satisfying. Perhaps another possibly better approach would be to provide a mockable component. But that complicates the programmer's API with more template parameters. I could keep going along this line but I won't.

Testability

I often struggle with how to make my classes testable. I sometimes find that the issue is hidden state. I then have to find ways to expose and manipulate that state in test cases. Sometimes that causes new problems typically exposing interfaces that exist solely for unit testing. Putting them in a separate public section helps but is less than satisfying. Perhaps another possibly better approach would be to provide a mockable component. But that complicates the programmer's API with more template parameters. I could keep going along this line but I won't.

Testability

I often struggle with how to make my classes testable. I sometimes find that the issue is hidden state. I then have to find ways to expose and manipulate that state in test cases. Sometimes that causes new problems typically exposing interfaces that exist solely for unit testing. Putting them in a separate public section helps but is less than satisfying. Perhaps another possibly better approach would be to provide a mockable component. But that complicates the programmer's API with more template parameters. I could keep going along this line but I won't.

Testability

- Unit Tests
- TDD

Testability: Hidden State

Testability: Hidden State

```
/**
 * The crux of this pattern relies on the fact that a template
 * specialization can be made with any member, even private members.
 * The biggest hurdle is that the access must be self-contained
 * within that particular specialization, which means that its use
 * is very limited.
 *
 * The code here uses mutable public static members, which are given
 * the pointer-to-member values in the specialization.
 *
 * This technique has been around in many forms since the original
 * C++ standard in 1998, but is personally attributed to a snippet
 * of code from Dave Abrahams.
 */
```

Testability: Hidden State

```
/**
 * The crux of this pattern relies on the fact that a template
 * specialization can be made with any member, even private members.
 * The biggest hurdle is that the access must be self-contained
 * within that particular specialization, which means that its use
 * is very limited.
 *
 * The code here uses mutable public static members, which are given
 * the pointer-to-member values in the specialization.
 *
 * This technique has been around in many forms since the original
 * C++ standard in 1998, but is personally attributed to a snippet
 * of code from Dave Abrahams.
 */
```

Testability: Hidden State

```
template <typename TagT, auto...>
struct Sudo;
```

Testability: Hidden State

```
template <typename TagT, auto...>
struct Sudo;

template <typename TagT>
struct Sudo<TagT>
{
   inline static typename TagT::type value{};
};
```

Testability: Hidden State

```
template <typename TagT, auto...>
struct Sudo;

template <typename TagT>
struct Sudo<TagT>
{
   inline static typename TagT::type value{};
};

template <typename TagT, typename TagT::type Member>
struct Sudo<TagT, Member>
{
    inline static const decltype(Member) value =
        Sudo<TagT>::value = Member;
};
```

Testability: Hidden State

```
template <typename TagT, auto...>
struct Sudo;

template <typename TagT>
struct Sudo<TagT>
{
   inline static typename TagT::type value{};
};

template <typename TagT, typename TagT::type Member>
struct Sudo<TagT, Member>
{
    inline static const decltype(Member) value =
        Sudo<TagT>::value = Member;
};
```

Testability: Hidden State

```
template <typename TagT, typename ObjectT>
decltype(auto)
sudo(ObjectT && object)
{
    assert(Sudo<TagT>::value);
    return std::invoke(
        Sudo<TagT>::value,
        std::forward<ObjectT>(object));
}
```

Testability: Hidden State

```
template <typename TagT, typename ObjectT>
decltype(auto)
sudo(ObjectT && object)
{
    assert(Sudo<TagT>::value);
    return std::invoke(
        Sudo<TagT>::value,
        std::forward<ObjectT>(object));
}

template <typename ClassT, typename MemberT, typename = void>
struct SudoTag
{
    using type = MemberT ClassT::*;
};
```

Testability: Hidden State

Testability: Hidden State

```
namespace foo::bar { class Blarg { int x = 42; }; }
```

Testability: Hidden State

```
namespace foo::bar { class Blarg { int x = 42; }; }

using Blarg_x = SudoTag<foo::bar::Blarg, int>;
```

Testability: Hidden State

```
namespace foo::bar { class Blarg { int x = 42; }; }

using Blarg_x = SudoTag<foo::bar::Blarg, int>;

template struct Sudo<Blarg_x, &foo::bar::Blarg::x>;
```

Testability: Hidden State

```
namespace foo::bar { class Blarg { int x = 42; }; }

using Blarg_x = SudoTag<foo::bar::Blarg, int>;

template struct Sudo<Blarg_x, &foo::bar::Blarg::x>;

int
main()
{
    auto blarg = foo::bar::Blarg{};
    std::cout << sudo<Blarg_x>(blarg) << '\n'; // print blarg.x
}
```

Testability: Hidden State

```
#define SUDO(NAMESPACE, TYPE, CLASS, MEMTYPE, MEMBER) \
    namespace NAMESPACE { \
    struct TYPE \
    : SudoTag<CLASS, MEMTYPE> \
    { }; \
    } \
    template struct Sudo<NAMESPACE::TYPE, &CLASS::MEMBER>

// Then, you can use it like this...
BP_SUDO(foo::bar, Blarg_x, foo::bar::Blarg, int, x);
```

Testability

Testability

Testability

Donald Knuth

Testability or Provability?

```
procedure insert2 (integer x, ℓ)
begin B[ℓ] ← B[ℓ] ∨ (2 ↑ (x mod 16));
    size[ℓ] ← size[ℓ]+1;
    if x < least[ℓ] then least[ℓ] ← x
    else if x > greatest[ℓ] then greatest[ℓ] ← x;
    end;
```

The implementation of deletion would be similar. It is safe to use 0 and $2^{16}-1$ for $-\infty$ and $+\infty$.

Beware of bugs in the above code; I have only proved it correct, not tried it.

https://staff.fnwi.uva.nl/p.vanemdeboas/knuthnote.pdf

Knuth on Unit Tests

In that vein, today’s developers frequently build programs writing small code increments followed by immediate compilation and the creation and running of unit tests. What are your thoughts on this approach to software development?

https://www.informit.com/articles/article.aspx?p=1193856

Knuth on Unit Tests

In that vein, today’s developers frequently build programs writing small code increments followed by immediate compilation and the creation and running of unit tests. What are your thoughts on this approach to software development?

As to your real question, immediate compilation, unit tests, extreme programming, and TDD combine to form the ultimate silver bullet. I've been waiting my entire career for such an advancement in computer programming methodology.

https://www.informit.com/articles/article.aspx?p=1193856

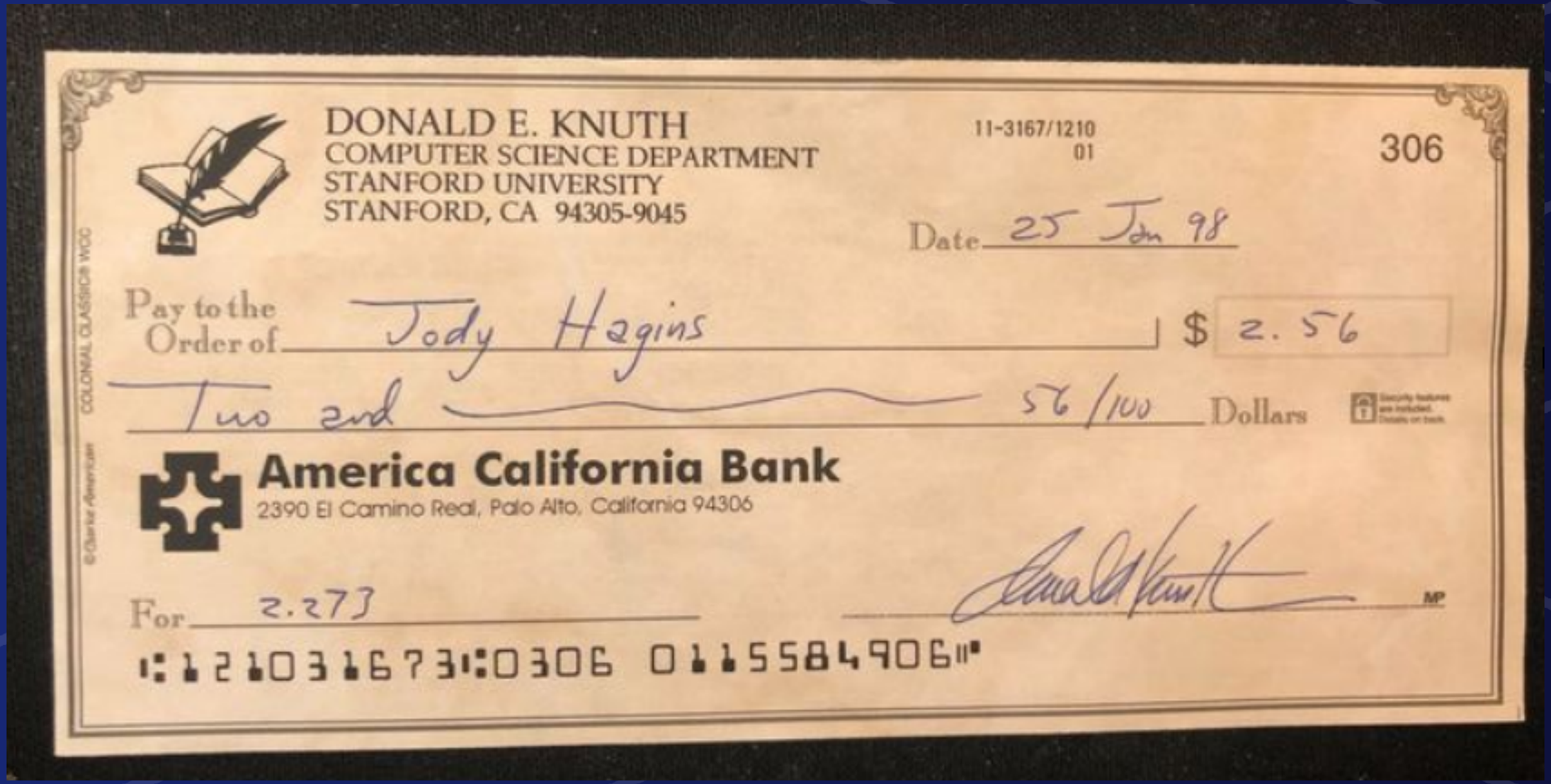
Knuth on Unit Tests

In that vein, today’s developers frequently build programs writing small code increments followed by immediate compilation and the creation and running of unit tests. What are your thoughts on this approach to software development?

As to your real question, the idea of immediate compilation and "unit tests" appeals to me only rarely, when I’m feeling my way in a totally unknown environment and need feedback about what works and what doesn’t. Otherwise, lots of time is wasted on activities that I simply never need to perform or even think about. Nothing needs to be "mocked up."

https://www.informit.com/articles/article.aspx?p=1193856

Testability



Knuth on ...

Can you give some examples that are currently in vogue, which developers shouldn’t adopt simply because they’re currently popular or because that’s the way they’re currently done? Would you care to identify important examples of this outside of software development?

https://www.informit.com/articles/article.aspx?p=1193856

Knuth on ...

I hate to duck your questions even though I also hate to offend other people's sensibilities—given that software methodology has always been akin to religion.

https://www.informit.com/articles/article.aspx?p=1193856

Knuth on ...

I hate to duck your questions even though I also hate to offend other people's sensibilities—given that software methodology has always been akin to religion. With the caveat that there's no reason anybody should care about the opinions of a computer scientist/mathematician like me regarding software development,

https://www.informit.com/articles/article.aspx?p=1193856

Knuth on Extreme Programming

I hate to duck your questions even though I also hate to offend other people's sensibilities—given that software methodology has always been akin to religion. With the caveat that there's no reason anybody should care about the opinions of a computer scientist/mathematician like me regarding software development, let me just say that almost everything I've ever heard associated with the term "extreme programming" sounds like exactly the wrong way to go...with one exception. The exception is the idea of working in teams and reading each other's code. That idea is crucial, and it might even mask out all the terrible aspects of extreme programming that alarm me.

https://www.informit.com/articles/article.aspx?p=1193856

Knuth on Extreme Programming and Reusable Code

I also must confess to a strong bias against the fashion for reusable code. To me, "re-editable code" is much, much better than an untouchable black box or toolkit. I could go on and on about this. If you're totally convinced that reusable code is wonderful, I probably won't be able to sway you anyway, but you'll never convince me that reusable code isn't mostly a menace.

https://www.informit.com/articles/article.aspx?p=1193856

Knuth on His Testing Methodology

I generally get best results by writing a test program that no sane user would ever think of writing. My test programs are intended to break the system, to push it to its extreme limits, to pile complication on complication, in ways that the system programmer never consciously anticipated. To prepare such test data, I get into the meanest, nastiest frame of mind that I can manage, and I write the cruelest code I can think of; then I turn around and embed that in even nastier constructions that are almost obscene.

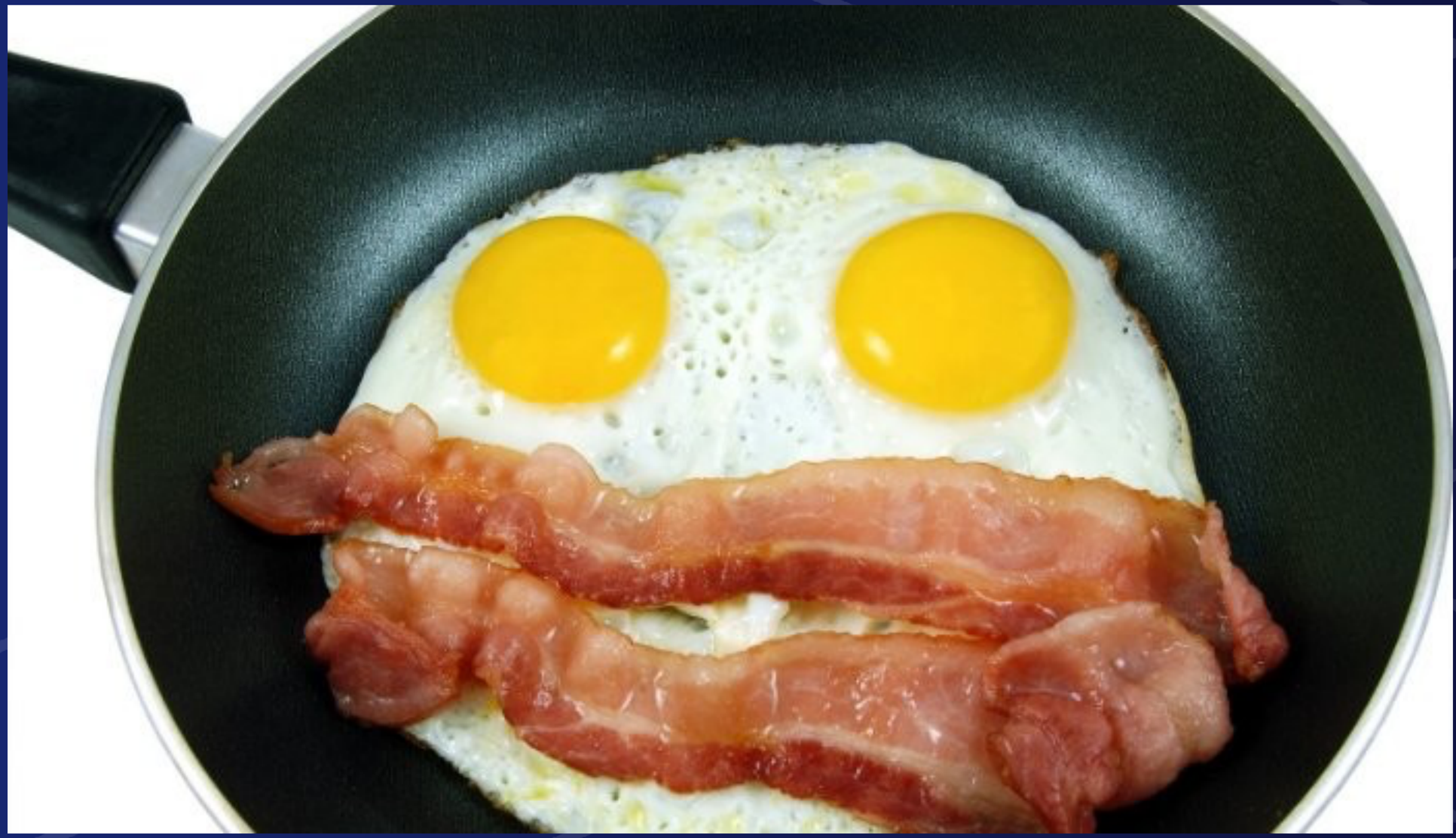
Donald Knuth, *The Errors of TEX* (1989)

How do you test your software?

How do you test your software?

https://en.wikipedia.org/wiki/Cruise_missile

Commitment to testing your software



https://www.linkedin.com/pulse/involved-vs-committed-you-pig-forrest

How do you test your software?

```
Blarg do_something(std::int64_t price, std::uint64_t quantity);
```

How do you test your software?

```
Blarg do_something(std::int64_t price, std::uint64_t quantity);
```

How do you test your software?

```
Blarg do_something(std::int64_t price, std::uint64_t quantity);
```

C++ Compiler: An Amazing Test Tool

```
Blarg do_something(Price price, Quantity quantity);
```

C++ Compiler: An Amazing Test Tool

```
Blarg do_something(Price price, Quantity quantity);
```

No Naked Types

C++ Compiler: An Amazing Test Tool

```
Blarg do_something(Price price, Quantity quantity);

Blarg do_something(Price, Quantity);
```

C++ Compiler: An Amazing Test Tool

```
Blarg do_something(
    Price bid_price,
    Quantity bid_quantity,
    Price ask_price,
    Quantity ask_quantity);
```

C++ Compiler: An Amazing Test Tool

```
Blarg do_something(
    BidPrice bid_price,
    BidQuantity bid_quantity,
    AskPrice ask_price,
    AskQuantity ask_quantity);
```

```
Blarg do_something(
    BidPrice,
    BidQuantity,
    AskPrice,
    AskQuantity);
```

Easy Strong Types???

```
struct [[clang::annotate("strong(int)")]] Flip;
struct [[clang::annotate("strong(int,+,-,->")]] Flop;
```

What do you test? What is an API?

The Hinnant Rule

compiler implicitly declares

user declares	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

The Hinnant Rule

https://www.youtube.com/watch?v=vLinb2fgkHk

What do you test? What is an API?

What do you test? What is an API?

Knight Capital Group

Knight

Company type	Subsidiary
Traded as	NYSE: KCG, until July 1, 2013
Industry	Financial services
Founded	1995
Fate	Acquired by Getco LLC in 2013, forming KCG Holdings
Headquarters	Jersey City, New Jersey, United States
Key people	Thomas Joyce, chairman and chief executive officer
Products	Market making and trading
Revenue	$1.404 billion USD (2011)
Net income	$115.2 million USD (2011)
Number of employees	1,418 (2012)
Website	www.knight.com

What do you test? What is an API?

$-10,000,000 per minute for 45 minutes

Knight Capital Cease and Desist

SECURITIES EXCHANGE ACT OF 1934
Release No. 70694 / October 16, 2013

ADMINISTRATIVE PROCEEDING
File No. 3-15570

In the Matter of

Knight Capital Americas LLC

Respondent.

ORDER INSTITUTING ADMINISTRATIVE AND CEASE-AND-DESIST PROCEEDINGS, PURSUANT TO SECTIONS 15(b) AND 21C OF THE SECURITIES EXCHANGE ACT OF 1934, MAKING FINDINGS, AND IMPOSING REMEDIAL SANCTIONS AND A CEASE-AND-DESIST ORDER

Summary

1. On August 1, 2012, Knight Capital Americas LLC ("Knight") experienced a significant error in the operation of its automated routing system for equity orders, known as SMARS. While processing 212 small retail orders that Knight had received from its customers, SMARS routed millions of orders into the market over a 45-minute period, and obtained over 4 million executions in 154 stocks for more than 397 million shares. By the time that Knight stopped sending the orders, Knight had assumed a net long position in 80 stocks of approximately $3.5 billion and a net short position in 74 stocks of approximately $3.15 billion. Ultimately, Knight lost over $460 million from these unwanted positions. The subject of these proceedings is Knight's violation of a Commission rule that requires brokers or dealers to have controls and procedures in place reasonably designed to limit the risks associated with their access to the markets, including the risks associated with automated systems and the possibility of these types of errors.

Retail Liquidity Program

12. To enable its customers' participation in the Retail Liquidity Program ("RLP") at the New York Stock Exchange,[5] which was scheduled to commence on August 1, 2012, Knight made a number of changes to its systems and software code related to its order handling processes. These changes included developing and deploying new software code in SMARS. SMARS is an automated, high speed, algorithmic router that sends orders into the market for execution. A core function of SMARS is to receive orders passed from other components of Knight's trading platform ("parent" orders) and then, as needed based on the available liquidity, send one or more representative (or "child") orders to external venues for execution.

General Idea: Replace Power Peg with RLP

13. Upon deployment, the new RLP code in SMARS was intended to replace unused code in the relevant portion of the order router. This unused code previously had been used for functionality called "Power Peg," which Knight had discontinued using many years earlier. Despite the lack of use, the Power Peg functionality remained present and callable at the time of the RLP deployment. The new RLP code also repurposed a flag that was formerly used to activate the Power Peg code. Knight intended to delete the Power Peg code so that when this flag was set to "yes," the new RLP functionality—rather than Power Peg—would be engaged.

General Idea: Replace Power Peg with RLP

13. Upon deployment, the new RLP code in SMARS was intended to replace unused code in the relevant portion of the order router. This unused code previously had been used for functionality called "Power Peg," which Knight had discontinued using many years earlier. Despite the lack of use, the Power Peg functionality remained present and callable at the time of the RLP deployment. The new RLP code also repurposed a flag that was formerly used to activate the Power Peg code. Knight intended to delete the Power Peg code so that when this flag was set to "yes," the new RLP functionality—rather than Power Peg—would be engaged.

General Idea: Replace Power Peg with RLP

13. Upon deployment, the new RLP code in SMARS was intended to replace unused code in the relevant portion of the order router. This unused code previously had been used for functionality called "Power Peg," which Knight had discontinued using many years earlier. Despite the lack of use, the Power Peg functionality remained present and callable at the time of the RLP deployment. The new RLP code also repurposed a flag that was formerly used to activate the Power Peg code. Knight intended to delete the Power Peg code so that when this flag was set to "yes," the new RLP functionality—rather than Power Peg—would be engaged.

Power Peg Routes Orders Until Filled

14. When Knight used the Power Peg code previously, as child orders were executed, a cumulative quantity function counted the number of shares of the parent order that had been executed. This feature instructed the code to stop routing child orders after the parent order had been filled completely. In 2003, Knight ceased using the Power Peg functionality. In 2005, Knight moved the tracking of cumulative shares function in the Power Peg code to an earlier point in the SMARS code sequence. Knight did not retest the Power Peg code after moving the cumulative quantity function to determine whether Power Peg would still function correctly if called.

Power Peg Value No Longer Used
Source Not Removed

14. When Knight used the Power Peg code previously, as child orders were executed, a cumulative quantity function counted the number of shares of the parent order that had been executed. This feature instructed the code to stop routing child orders after the parent order had been filled completely. In 2003, Knight ceased using the Power Peg functionality. In 2005, Knight moved the tracking of cumulative shares function in the Power Peg code to an earlier point in the SMARS code sequence. Knight did not retest the Power Peg code after moving the cumulative quantity function to determine whether Power Peg would still function correctly if called.

Rewrite; Power Peg Moved, Not Removed

14. When Knight used the Power Peg code previously, as child orders were executed, a cumulative quantity function counted the number of shares of the parent order that had been executed. This feature instructed the code to stop routing child orders after the parent order had been filled completely. In 2003, Knight ceased using the Power Peg functionality. In 2005, Knight moved the tracking of cumulative shares function in the Power Peg code to an earlier point in the SMARS code sequence. Knight did not retest the Power Peg code after moving the cumulative quantity function to determine whether Power Peg would still function correctly if called.

Did Not Retest Power Peg

14. When Knight used the Power Peg code previously, as child orders were executed, a cumulative quantity function counted the number of shares of the parent order that had been executed. This feature instructed the code to stop routing child orders after the parent order had been filled completely. In 2003, Knight ceased using the Power Peg functionality. In 2005, Knight moved the tracking of cumulative shares function in the Power Peg code to an earlier point in the SMARS code sequence. Knight did not retest the Power Peg code after moving the cumulative quantity function to determine whether Power Peg would still function correctly if called.

Deploy Version to Support RLP

15. Beginning on July 27, 2012, Knight deployed the new RLP code in SMARS in stages by placing it on a limited number of servers in SMARS on successive days. During the deployment of the new code, however, one of Knight's technicians did not copy the new code to one of the eight SMARS computer servers. Knight did not have a second technician review this deployment and no one at Knight realized that the Power Peg code had not been removed from the eighth server, nor the new RLP code added. Knight had no written procedures that required such a review.

Manual Deployment Mistake

15. Beginning on July 27, 2012, Knight deployed the new RLP code in SMARS in stages by placing it on a limited number of servers in SMARS on successive days. During the deployment of the new code, however, one of Knight's technicians did not copy the new code to one of the eight SMARS computer servers. Knight did not have a second technician review this deployment and no one at Knight realized that the Power Peg code had not been removed from the eighth server, nor the new RLP code added. Knight had no written procedures that required such a review.

No Review or Test of Deployment

15. Beginning on July 27, 2012, Knight deployed the new RLP code in SMARS in stages by placing it on a limited number of servers in SMARS on successive days. During the deployment of the new code, however, one of Knight's technicians did not copy the new code to one of the eight SMARS computer servers. Knight did not have a second technician review this deployment and no one at Knight realized that the Power Peg code had not been removed from the eighth server, nor the new RLP code added. Knight had no written procedures that required such a review.

RLP Goes Live on 7 Servers

16. On August 1, Knight received orders from broker-dealers whose customers were eligible to participate in the RLP. The seven servers that received the new code processed these orders correctly. However, orders sent with the repurposed flag to the eighth server triggered the defective Power Peg code still present on that server. As a result, this server began sending child orders to certain trading centers for execution. Because the cumulative quantity function had been moved, this server continuously sent child orders, in rapid sequence, for each incoming parent order without regard to the number of share executions Knight had already received from trading centers. Although one part of Knight's order handling system recognized that the parent orders had been filled, this information was not communicated to SMARS.

Power Peg Runs on 1 Server

16. On August 1, Knight received orders from broker-dealers whose customers were eligible to participate in the RLP. The seven servers that received the new code processed these orders correctly. However, orders sent with the repurposed flag to the eighth server triggered the defective Power Peg code still present on that server. As a result, this server began sending child orders to certain trading centers for execution. Because the cumulative quantity function had been moved, this server continuously sent child orders, in rapid sequence, for each incoming parent order without regard to the number of share executions Knight had already received from trading centers. Although one part of Knight's order handling system recognized that the parent orders had been filled, this information was not communicated to SMARS.

2005 Bites Back

16. On August 1, Knight received orders from broker-dealers whose customers were eligible to participate in the RLP. The seven servers that received the new code processed these orders correctly. However, orders sent with the repurposed flag to the eighth server triggered the defective Power Peg code still present on that server. As a result, this server began sending child orders to certain trading centers for execution. Because the cumulative quantity function had been moved, this server continuously sent child orders, in rapid sequence, for each incoming parent order without regard to the number of share executions Knight had already received from trading centers. Although one part of Knight's order handling system recognized that the parent orders had been filled, this information was not communicated to SMARS.

Why Should I Tell You?

16. On August 1, Knight received orders from broker-dealers whose customers were eligible to participate in the RLP. The seven servers that received the new code processed these orders correctly. However, orders sent with the repurposed flag to the eighth server triggered the defective Power Peg code still present on that server. As a result, this server began sending child orders to certain trading centers for execution. Because the cumulative quantity function had been moved, this server continuously sent child orders, in rapid sequence, for each incoming parent order without regard to the number of share executions Knight had already received from trading centers. Although one part of Knight's order handling system recognized that the parent orders had been filled, this information was not communicated to SMARS.

$460,000,000 in 45 Minutes

17. The consequences of the failures were substantial. For the 212 incoming parent orders that were processed by the defective Power Peg code, SMARS sent millions of child orders, resulting in 4 million executions in 154 stocks for more than 397 million shares in approximately 45 minutes. Knight inadvertently assumed an approximately $3.5 billion net long position in 80 stocks and an approximately $3.15 billion net short position in 74 stocks. Ultimately, Knight realized a $460 million loss on these positions.

What is an Error?

19. On August 1, Knight also received orders eligible for the RLP but that were designated for pre-market trading.[6] SMARS processed these orders and, beginning at approximately 8:01 a.m. ET, an internal system at Knight generated automated e-mail messages (called "BNET rejects") that referenced SMARS and identified an error described as "Power Peg disabled." Knight's system sent 97 of these e-mail messages to a group of Knight personnel before the 9:30 a.m. market open. Knight did not design these types of messages to be system alerts, and Knight personnel generally did not review them when they were received. However,

Just a Notification: Nothing to See

19. On August 1, Knight also received orders eligible for the RLP but that were designated for pre-market trading.[6] SMARS processed these orders and, beginning at approximately 8:01 a.m. ET, an internal system at Knight generated automated e-mail messages (called "BNET rejects") that referenced SMARS and identified an error described as "Power Peg disabled." Knight's system sent 97 of these e-mail messages to a group of Knight personnel before the 9:30 a.m. market open. Knight did not design these types of messages to be system alerts, and Knight personnel generally did not review them when they were received. However,

If Only Someone Told Me

these messages were sent in real time, were caused by the code deployment failure, and provided Knight with a potential opportunity to identify and fix the coding issue prior to the market open. These notifications were not acted upon before the market opened and were not used to diagnose the problem after the open.

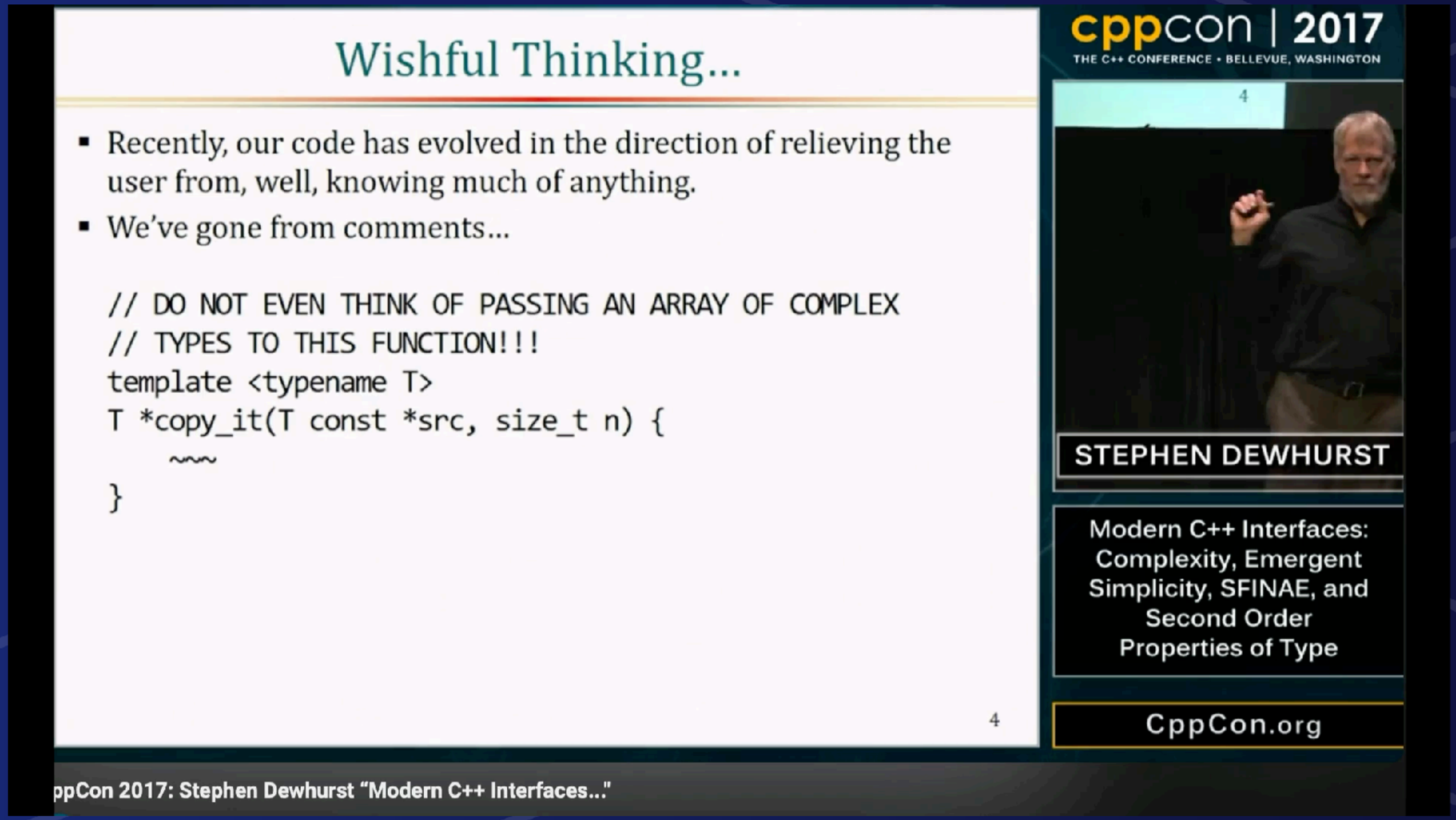
No Automated Monitoring

24. On the morning of August 1, the 33 Account began accumulating an unusually large position resulting from the millions of executions of the child orders that SMARS was sending to the market. Because Knight did not link the 33 Account to pre-set, firm-wide capital thresholds that would prevent the entry of orders, on an automated basis, that exceeded those thresholds, SMARS continued to send millions of child orders to the market despite the fact that the parent orders already had been completely filled.[7] Moreover, because the 33 Account held

Rollback???

27. On August 1, Knight did not have supervisory procedures concerning incident response. More specifically, Knight did not have supervisory procedures to guide its relevant personnel when significant issues developed. On August 1, Knight relied primarily on its technology team to attempt to identify and address the SMARS problem in a live trading environment. Knight's system continued to send millions of child orders while its personnel attempted to identify the source of the problem. In one of its attempts to address the problem, Knight uninstalled the new RLP code from the seven servers where it had been deployed correctly. This action worsened the problem, causing additional incoming parent orders to activate the Power Peg code that was present on those servers, similar to what had already occurred on the eighth server.

Really Good Bugs



How can we test it?

How can we test it?

- Remove old code and add new code at the same time
- Reuse an existing "value" for completely different functionality
- Stopped using Power Peg in 2003, dead code still around
- Refactored code in 2005 ; moved dead code ; test still pass
- Manual deployment ; no review ; no tests
- Different components had different view of same thing
- email for important log messages
- Rollback made matters worse

How can we test it?

- Remove old code and add new code at the same time
- Reuse an existing "value" for completely different functionality
- Stopped using Power Peg in 2003, dead code still around
- Refactored code in 2005 ; moved dead code ; test still pass
- Manual deployment ; no review ; no tests
- Different components had different view of same thing
- email for important log messages
- Rollback made matters worse

How can we test it?

- Remove old code and add new code at the same time
  - Separate tasks into smaller chunks

How can we test it?

- Remove old code and add new code at the same time
  - Separate tasks into smaller chunks
  - Static analysis tools

How can we test it?

- Remove old code and add new code at the same time
  - Separate tasks into smaller chunks
  - Static analysis tools
  - Formal Design

How can we test it?

- Remove old code and add new code at the same time
  - Separate tasks into smaller chunks
  - Static analysis tools
  - Formal Design
  - Code Review

How can we test it?

- Remove old code and add new code at the same time
- Reuse an existing "value" for completely different functionality
- Stopped using Power Peg in 2003, dead code still around
- Refactored code in 2005 ; moved dead code ; test still pass
- Manual deployment ; no review ; no tests
- Different components had different view of same thing
- email for important log messages
- Rollback made matters worse

How can we test it?

- Reuse an existing "value" for completely different functionality

How can we test it?

- Reuse an existing "value" for completely different functionality
- Um... Don't do that.

How can we test it?

- Reuse an existing "value" for completely different functionality
- Um... Don't do that.
  - Code Review

How can we test it?

- Reuse an existing "value" for completely different functionality
- Um... Don't do that.
  - Code Review
  - Automated Tooling - part of CI

How can we test it?

- Reuse an existing "value" for completely different functionality
- Um...
  - Coc
  - Aut

That is good advice, but I don't think that would have prevented their problem.

They didn't have multiple values. They renamed an existing value. For example, what they did was like having an enum where POWER_PEG = 3 and they just renamed it to something like NYSE_RLP = 3.

Thus, the previous definition no longer existed. How do you prevent this kind of renaming?

How can we test it?

1. Historical Audit of Enum Values (Version Control Tracking)
2. Enumeration/Constant Versioning and Deprecation
3. Automated Static Code Analysis for Renaming
4. Semantic Versioning for Enums
5. Refactor Scripts and Tools to Validate Renaming
6. Pre-Deployment and Regression Testing
7. Enumeration Documentation and Usage Comments
8. Detect Renaming Through Code Review Process
9. Symbolic Constant Tracking

How can we test it?

1. Historical Audit of Enum Values (Version Control Tracking)
2. Enumeration/Constant Versioning and Deprecation
3. Automated Static Code Analysis for Renaming
4. Semantic Versioning for Enums

I'm not a java programmer, can you restate using C++

7. Enumeration Documentation and Usage Comments
8. Detect Renaming Through Code Review Process
9. Symbolic Constant Tracking

How can we test it?

- Reuse an existing "value" for completely different functionality
- Um... Don't do that.
  - Code Review
  - Automated Tooling - part of CI

write a program using libclang that takes two files as input and checks to see if the two files define the same enumeration, and if they do, then check to see that all enum names and values are unique between the two

How can we test it?

- Reuse an existing "value" for completely different functionality
- Um... Don't do that.
  - Code Review
  - Automated Tooling - part of CI

use rapidcheck to write a complete set of property based tests

How can we test it?

- Remove old code and add new code at the same time
- Reuse an existing "value" for completely different functionality
- Stopped using Power Peg in 2003, dead code still around
- Refactored code in 2005 ; moved dead code ; test still pass
- Manual deployment ; no review ; no tests
- Different components had different view of same thing
- email for important log messages
- Rollback made matters worse

How can we test it?

- Stopped using Power Peg in 2003, dead code still around
- Dead code must be removed

How can we test it?

- Stopped using Power Peg in 2003, dead code still around
- Dead code must be removed
  - Deprecation Policies

How can we test it?

- Stopped using Power Peg in 2003, dead code still around
- Dead code must be removed
  - Deprecation Policies
  - Code Review

How can we test it?

- Stopped using Power Peg in 2003, dead code still around
- Dead code must be removed
  - Deprecation Policies
  - Code Review
  - Static Analysis and Code Coverage

How can we test it?

- Remove old code and add new code at the same time
- Reuse an existing "value" for completely different functionality
- Stopped using Power Peg in 2003, dead code still around
- Refactored code in 2005 ; moved dead code ; test still pass
- Manual deployment ; no review ; no tests
- Different components had different view of same thing
- email for important log messages
- Rollback made matters worse

How can we test it?

- Refactored code in 2005 ; moved dead code ; test still pass
  - Deprecation Policies
  - Code Review
  - Static Analysis and Code Coverage
  - Functionality Tests (unit tests or Knuth's torture tests)

How can we test it?

- Remove old code and add new code at the same time
- Reuse an existing "value" for completely different functionality
- Stopped using Power Peg in 2003, dead code still around
- Refactored code in 2005 ; moved dead code ; test still pass
- Manual deployment ; no review ; no tests
- Different components had different view of same thing
- email for important log messages
- Rollback made matters worse

How can we test it?

- Manual deployment ; no review ; no tests
  - NEVER do manual deployments
  - All deployments must be automated
  - Comprehensive system tests after deployment in real environment
    - Or as close as possible (cruise missile tests)
  - Reviews are enforced by CI tools
  - Can have deployment require a tool where multiple people ack

How can we test it?

- Remove old code and add new code at the same time
- Reuse an existing "value" for completely different functionality
- Stopped using Power Peg in 2003, dead code still around
- Refactored code in 2005 ; moved dead code ; test still pass
- Manual deployment ; no review ; no tests
- Different components had different view of same thing
- email for important log messages
- Rollback made matters worse

How can we test it?

- Different components had different view of same thing
  - Monitor and SMARS had different cumqty
    - Validate different system views
    - Event Logs

How can we test it?

- Remove old code and add new code at the same time
- Reuse an existing "value" for completely different functionality
- Stopped using Power Peg in 2003, dead code still around
- Refactored code in 2005 ; moved dead code ; test still pass
- Manual deployment ; no review ; no tests
- Different components had different view of same thing
- email for important log messages
- Rollback made matters worse

How can we test it?

- email for important log messages
  - Policy
  - All logs should be monitored with automated processes
    - Can't generate lazy logs - need to be structured

How can we test it?

- Remove old code and add new code at the same time
- Reuse an existing "value" for completely different functionality
- Stopped using Power Peg in 2003, dead code still around
- Refactored code in 2005 ; moved dead code ; test still pass
- Manual deployment ; no review ; no tests
- Different components had different view of same thing
- email for important log messages
- Rollback made matters worse

How can we test it?

- Rollback made matters worse
  - Must do full system tests including rollbacks
  - Must rollback and entire system - not just parts
  - Must be included in comprehensive off-hours testing with real system

Generated Code

- Got your own code generator?
  - How do you test it?
    - How do you test the generated code?
    - How do you code review the generated code?
  - Always commit code generated from your own generator into the source repository
    - I know everyone else says not to do that
      - I used to be part of everyone...
      - Everyone else is wrong
    - I learned the hard way - not as hard as Knight, but hard enough to never repeat that mistake again.