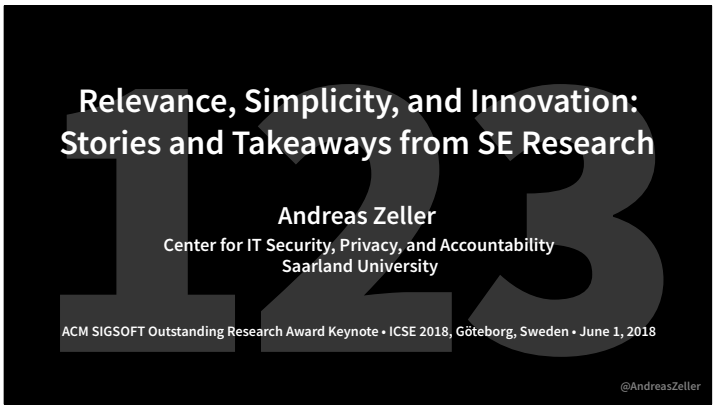


Relevance, Simplicity, and Innovation: Stories and Takeaways from Software Engineering Research

ACM SIGSOFT Outstanding Research Award Keynote • ICSE 2018, Göteborg, Sweden • June 1, 2018

Andreas Zeller, CISPA / Saarland University

Abstract. The year is 1993, and I give my very first talk at a big software engineering conference. Right in the middle of my example, a professor stands up and exclaims with a mocking smile “To me, this looks like a solution looking for a problem!”. The audience erupts in laughter, and my advisor sits in the first row, grinning. How would I get out of there? And why would this experience shape all of my career from now? Telling three stories around three conference events, I unfold lessons on impact in software engineering research: Do relevant work – strive for simplicity – keep on innovating.

Thank you very much, everyone. I know this has been a great conference, but now you're eager to get to get to your planes, to get back to your offices, to get back to friends and family. So in the next couple of minutes,

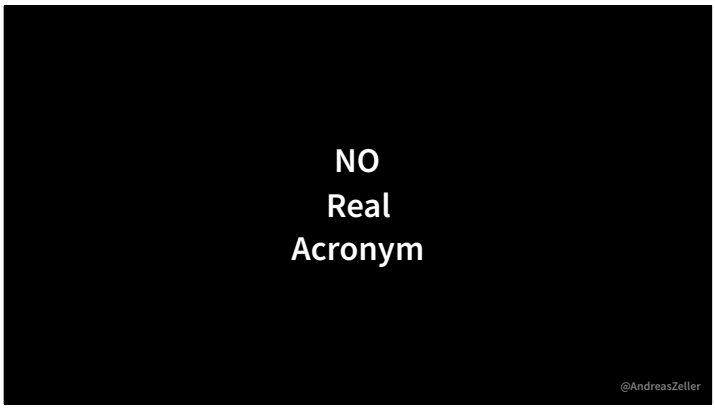
I am going to restrict myself to three short stories. Not more, not less. All three of them are connected to some conference talk, so I guess I'm in the right place to share them.

All these stories revolve around talks at conferences, and here's my first one, almost 25 years ago. This is in 1993

at the German national conference for Software Engineering, in Dortmund, Germany. Anyone from Dortmund, here? At this time, I am a PhD student, and this

Is my first talk ever.

I am presenting an experimental programming environment called NORA. NORA stands



for no real acronym, so it's pretty generic, but what this is about is actually

Theorem Provers in SE

@AndreasZeller

one of the first uses of theorem provers in Software Engineering.

Configuration Management with Feature Logic

@AndreasZeller

My own topic would be configuration management with feature logic, using features to represent variability and changes

---

Component Search

@AndreasZeller

Our key example that day, however, would be component search.

Find a sorting function

@AndreasZeller

The idea is that you'd have a huge library of components, and you'd be able to find a sorting function

---

using postconditions

@AndreasZeller

by specifying the pre- and postconditions of the function you're searching for.

$$\forall i, j: i < j \Rightarrow a'[i] \leq a'[j]$$

@AndreasZeller

So, here's the postcondition. You want the resulting array a' to be sorted.

$$\forall i, j: i < j \Rightarrow a'[i] \leq a'[j]$$
$$\wedge\ \forall x \in a \cup a': |\{i: x = a[i]\}| = |\{j: x = a'[j]\}|$$

@AndreasZeller

But then, the output array also has to be a permutation of the input array, so you have to add that little extra. So, after entering all this, I was showing how our system would now retrieve the sorting function, when right in the middle of my talk, a guy stands up and shouts

---

"When I search a sorting function, I do *grep sort*"

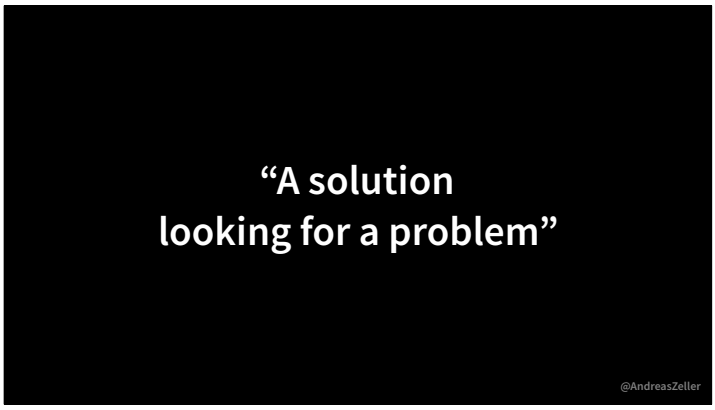
@AndreasZeller

"When I search for a sorting function, I do grep sort!" – to the great laughter of all attendees, maybe 100–150 people.

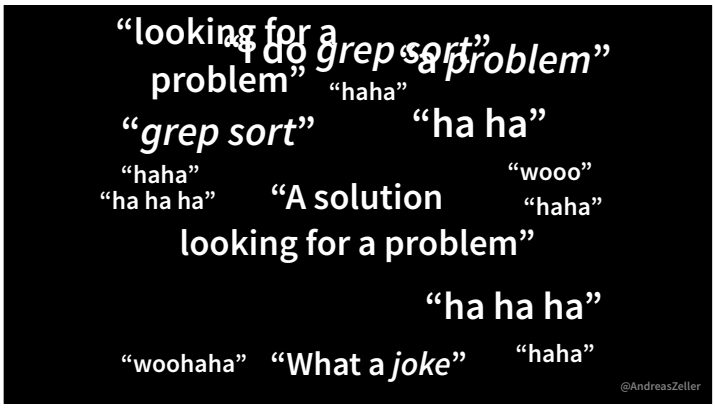
$$\forall i, j: i < j \Rightarrow a'[i] \leq a'[j]$$
$$\wedge\ \forall x \in a \cup a': |\{i: x = a[i]\}| = |\{j: x = a'[j]\}|$$

@AndreasZeller

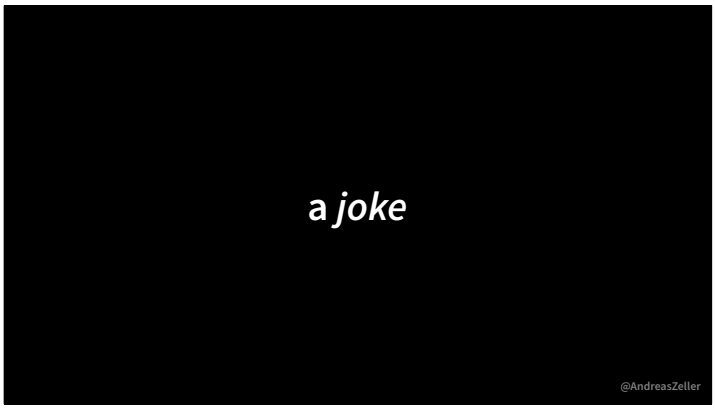
I look to my advisor, he's sitting in the first row, crossing his arms and grinning: How would I get out of that? So, I explain that this of course would not be sorting alone, you could even find a sorting function when you did not even have a name for sorting, and I restart – when another guy pops up and shouts:



"You know, to me this looks like a solution looking for a problem"



This closes it. I am done; I go through the remaining slides, but nobody listens anymore, and for the rest of the day, there's people laughing and pointing when they see me,



and I am eager to get the train home. All the way back, I am still enraged.

So that was the story of my first talk. Is the story over yet? Not quite. There's a couple of ways I can spin the remainder of the story.

I could tell how after being utterly devastated, I finally managed to find my path, and still make a great career in computer science. Guys, girls – don't listen to what old white farts say, follow your dreams, and in the end, you'll get ~~married and have many children~~ tenured and have many papers accepted.

I could also spin this from the angle of how important and ubiquitous theorem provers are today, how all of verification, testing, analysis depends on constraint solvers, model checkers, you name it. We were among the first, and today, I am right.

Relevance

@AndreasZeller

But the spin I'd like to give this story is yet another one, namely the question of relevance. Today, when I think back of this story,

They were right

@AndreasZeller

It turns out that these guys shouting into my talk were right all along. Think of how programmers work when they search for some function.

Google

@AndreasZeller

They go to Google

StackOverflow

@AndreasZeller

They go to StackOverflow

---

grep sort is everywhere

@AndreasZeller

So much of programming is searching today. It is "grep sort" everywhere.

They were right

@AndreasZeller

So, the essence of the story is that they were right all along – developers want simple tools that work, not some made-up formalism that only PhD students understand.

DDD

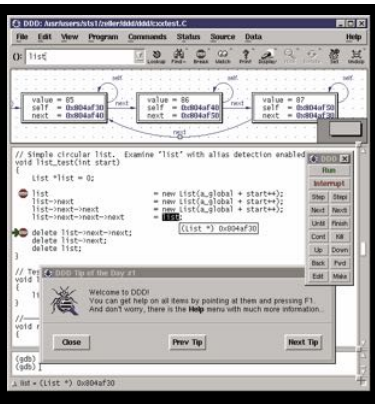
@AndreasZeller

One important consequence for me was that I started a sideline these days, together with a student of mine, Dorothea Lütkehaus. Already in my master's thesis, I had built a library that could visualize data structures. We thought of building this into a debugger, and built a tool, called DDD

Data Display Debugger

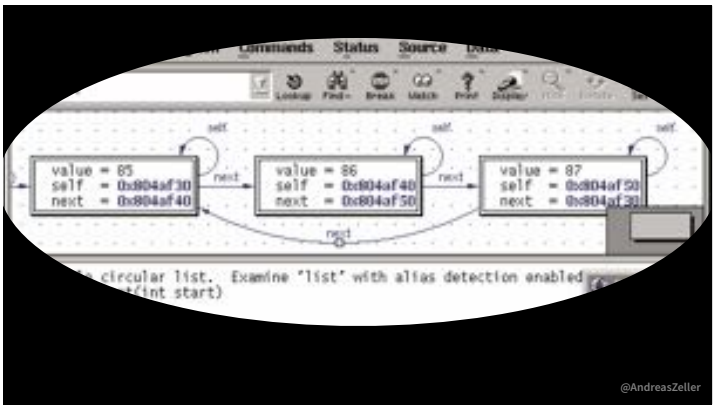
@AndreasZeller

for Data Display Debugger



@AndreasZeller

This is it



And at the top, you can see DDD nicely visualizing a linked list

---

DDD

@AndreasZeller

Now, it turned out that DDD was among the first debuggers with a decent graphical user interface. People loved it.

GNU DDD

@AndreasZeller

It became a GNU program – I got a nice letter by Richard Stallman –

postcards

@AndreasZeller

– and developers from all over the world sent in postcards to thank us for making it available.

---

postcards vs. citations

@AndreasZeller

These postcards at the time were far more important to me than citations I would get. People were actually *using* my stuff.

---

tool vs. paper

@AndreasZeller

This is because DDD was a *tool* that would get things done, with immediate usefulness.

concrete vs. abstract

@AndreasZeller

A concrete benefit, not just some abstract concept that may or may not be adopted.

---

useful

@AndreasZeller

The key metric here is usefulness. DDD was clearly useful. And this usefulness was

---

useful = better

@AndreasZeller

that made it better than the state of practice. Usefulness is the key metric in Software Engineering, so this experience prompted me to ask questions like

Is my research useful?

@AndreasZeller

Is my research useful?

---

for whom?

@AndreasZeller

And for whom?

What do developers need?

@AndreasZeller

What is it that developers – our customers, our key audience – actually need?

where I was

where I should be

@AndreasZeller

So here I was with my research, feature logic, theorem provers – but I felt out of place.

What do developers need?

@AndreasZeller

What is it that developers actually need? We can ask them.

Analyze This! 145 Questions for Data Scientists in Software Engineering

Andrew Begel
Microsoft Research
Redmond, WA, USA
Andrew.Begel@microsoft.com

Thomas Zimmermann
Microsoft Research
Redmond, WA, USA
tzimmer@microsoft.com

ABSTRACT
In this paper, we present the results from two surveys related to data science applied to software engineering. The first survey solicited questions that software engineers would like data scientists to investigate about software, about software processes and practices, and about software engineers. Our analyses resulted in a list of 145 questions grouped into 12 categories. The second survey asked a different pool of software engineers to rate these 145 questions and identify the most important ones to work on first. Respondents favored questions that focus on how customers typically use their applications. We also saw opposition to questions that assess the performance of individual employees or compare them with one another. Our categorization and catalog of 145 questions can help researchers, practitioners, and educators to more easily focus their efforts on topics that are important to the software industry.

Categories and Subject Descriptors: D.2.9 [Management]

General Terms: Management, Human factors, Measurement

"I'm giving a talk on Monday to a room full of software engineering researchers who are specialists in data-mining software repositories (among other things). If you could get them to tackle any questions at all (well, any related to software or software development), what would you want them to do, and why?"

In an introduction to an empirical software engineering panel at ESEC/FSE 2013, Bertrand Meyer emphasized the need for the software engineering community to become more data-driven, and to "shed folkloric advice and anecdotal evidence." He presented a list of 11 questions "crying for evidence" [10] whose answers should be empirical, credible, and useful. By useful, Meyer meant, "providing answers to *questions of interest to practitioners*."

In this paper, we present a ranked list of questions that software engineers want to have answered by data scientists. The list was compiled from two surveys that we deployed among professional software engineers at Microsoft (Section 3).

1. In the first survey, we asked a random sample of 1,500 Microsoft engineers a question similar to Greg Wilson's. We

@AndreasZeller

* Do people ever write loop invariants? Does it help?
* How do we measure the productivity of our engineers?
* How do users typically use my application?

There's this extremely nice survey by Andy Begel and Tom Zimmermann at Microsoft from ICSE 2014, including questions such as the above

recommenders?

@AndreasZeller

Now, the word "recommender" does not occur in that paper.

---

models?

@AndreasZeller

Nor does the word "model" occur.

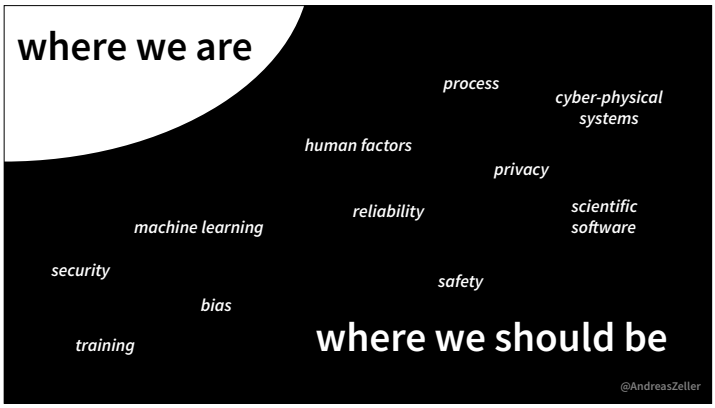
---

repair?

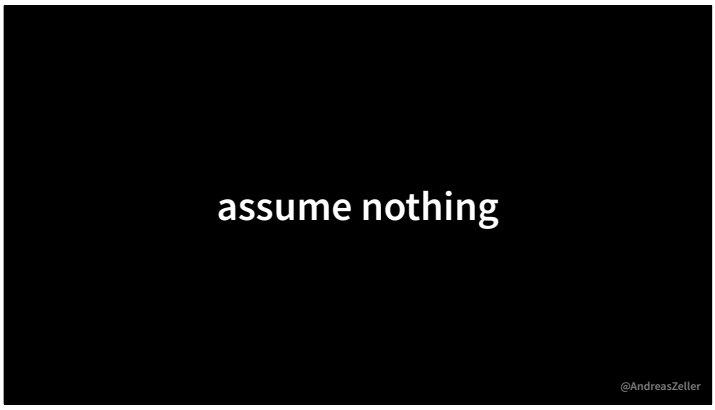
@AndreasZeller

Nor "repair".

Remember this slide, when I had my doubts?



I think that given the number of problems we are facing today – or still facing after all these years – we still are very much where the light is bright, where we know our strengths. Yet, maybe, we should venture out a bit more into the darkness. Talk to developers, talk to industry, find out where the real challenges are – and face them.



But when talking to developers, do not assume they will change anything because of you. They will not adopt your formal method just because you say so.

pave a way

@AndreasZeller

Make sure that they can adopt your approach with minimal effort. And pave a way toward this transition.

---

paper culture

@AndreasZeller

And here, I am not sure whether our paper-centric culture is the best way to achieve this. You are literally asking the reader to rebuild everything you describe.

---

tool vs. paper

@AndreasZeller

Actually, I think that tools are a much better way to achieve impact and relevance.

tool *and* paper

@AndreasZeller

And even better, I think that we should go and bring both together. I'd like to show an example.

Jupyter Notebook

@AndreasZeller

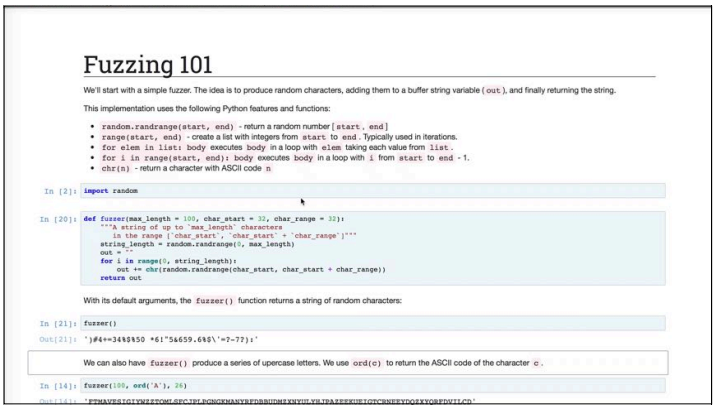
You may have heard of the Jupyter Notebook

Jupyter Notebook

ACM Software System Award 2018

@AndreasZeller

And if you haven't – they just got the software system award from ACM



Fuzzing 101

We'll start with a simple fuzzer. The idea is to produce random characters, adding them to a buffer string variable `out`, and finally returning the string.

This implementation uses the following Python features and functions:

- `random.randrange(start, end)` - return a random number `[start, end]`
- `range(start, end)` - create a list with integers from `start` to `end`. Typically used in iterations.
- `for elem in list: body` executes `body` in a loop with `elem` taking each value from `list`.
- `for i in range(start, end): body` executes `body` in a loop with `i` from `start` to `end` - 1.
- `chr(n)` - return a character with ASCII code `n`

```
In [2]: import random
```

```
In [20]: def fuzzer(max_length = 100, char_start = 32, char_range = 32):
             """A string of up to `max_length` characters
                in the range [`char_start`, `char_start` + `char_range`)"""
             string_length = random.randrange(0, max_length)
             out = ""
             for i in range(0, string_length):
                 out += chr(random.randrange(char_start, char_start + char_range))
             return out
```

With its default arguments, the `fuzzer()` function returns a string of random characters:

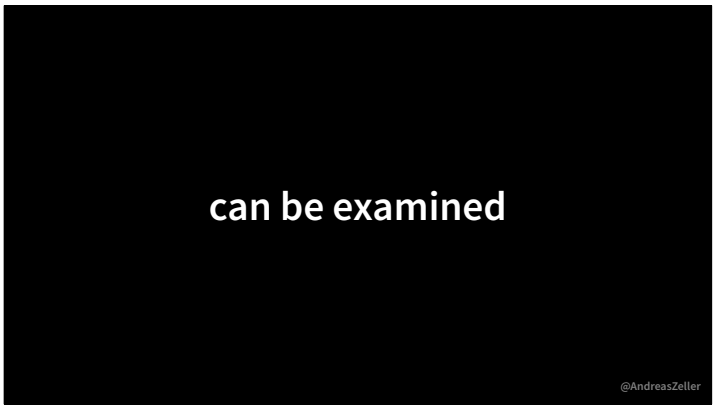
```
In [21]: fuzzer()
Out[21]: ')#4+=34%$%50 *6!"5&659.6%$\'=?~?7):'
```

We can also have `fuzzer()` produce a series of upercase letters. We use `ord(c)` to return the ASCII code of the character `c`.

```
In [14]: fuzzer(100, ord('A'), 26)
Out[14]: 'FTMAVESIGIYWZZTOMLSFCJBLRGNGXMANYRFDBSUDHZXNYULYHJPAZEEKUEIGTCRNEEYDQZXYORFDVILCD'
```

With Jupyter, you have a mix of text and code; you can edit both, and you can execute the code to immediately see the results of your actions. You have math typesetting, you have plots, you even have interactive elements such as sliders. All updated in real time.
Now, this is great for teaching, for students, for instructors. But if *this* were a scientific paper, what would it mean?

You could have small programs that embody the gist of your algorithm.
Small programs



that can be examined

can be experimented with

@AndreasZeller

that can be assessed and experimented with

---

can be taught

@AndreasZeller

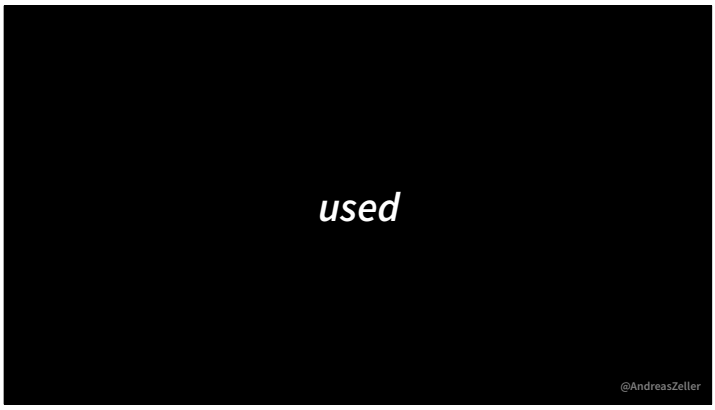
that can be taught

---

can be used

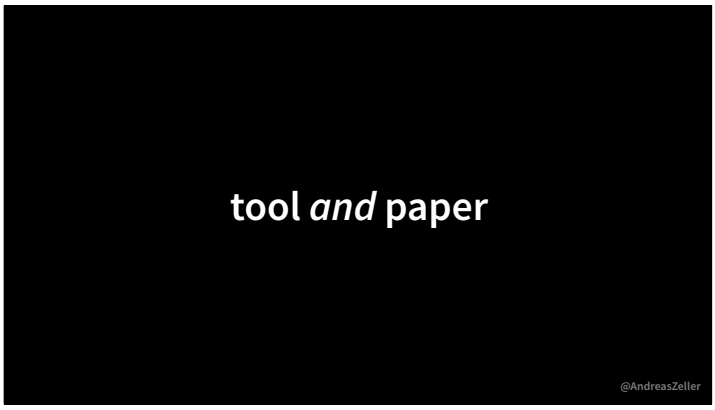
@AndreasZeller

that can be *used* by others –



used

and reused.



You'd have both: the tool and the paper.

Actually, why still have paper?

That was my first story – on relevance.

My second story is on simplicity. Six years later, it is 1999

And we are at ESEC/FSE, Toulouse, France

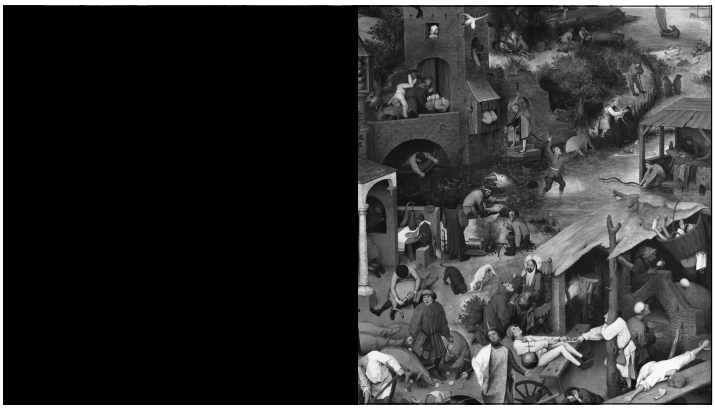
I have completed my PhD on version control

and the experience with DDD had raised my interest in debugging.

So I had come up with an idea that combines both: Version control and debugging

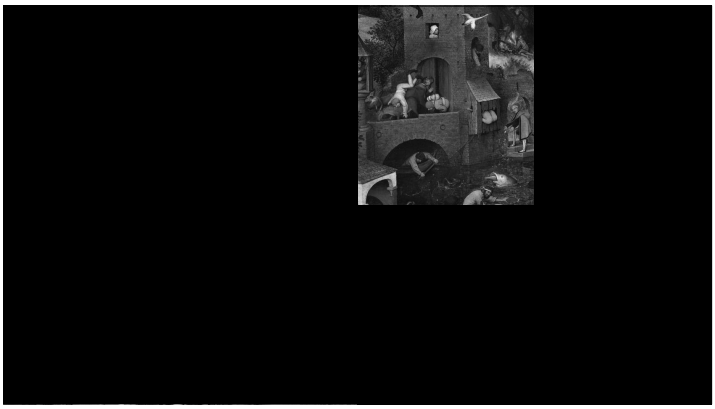
The core idea of delta debugging is very simple. You have a big set of possible influences (here's one big set of things), and in there, there's a small set that causes what you're looking for.

You can test, though, whether what you're searching for is in the set. So you try out one half



And another half. Turns out the cause is in here, so you keep it.

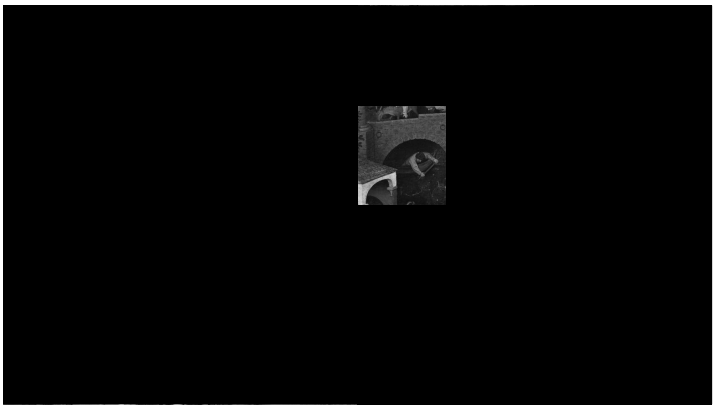
You repeat the process. Remove one half – hey, the effect is still there.



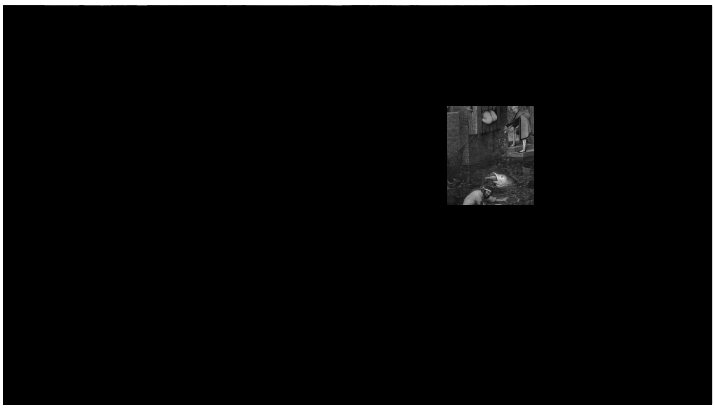
Again

And again

Turns out that now, it's in the other half



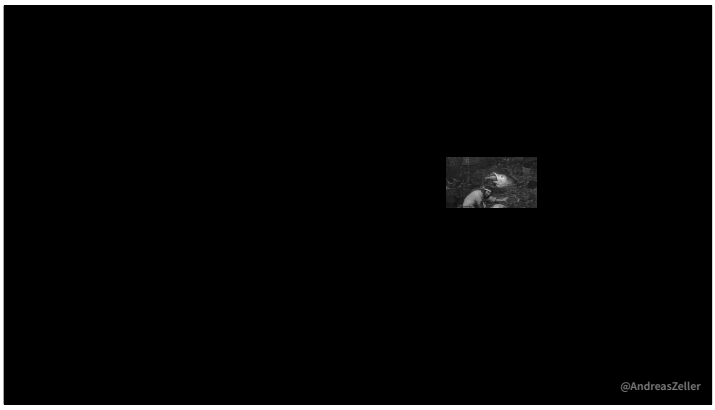
You keep on narrowing



And narrowing

And narrowing

And narrowing further. That's a bit small, right?



Okay, we'll go and enlarge things

This is what a process like delta debugging finds – the small subset that causes the bug. Can be in your input, in your version history, in your configuration

And it's typically a very tiny element or difference

in a big, big set.

```
def dd(c_pass, c_fail):
    n = 2
    while True:
        delta = listminus(c_fail, c_pass)
        deltas = split(delta, n); offset = 0; j = 0
        while j < n:
            i = (j + offset) % n
            next_c_pass = listunion(c_pass, deltas[i])
            next_c_fail = listminus(c_fail, deltas[i])
            if test(next_c_fail) == FAIL and n == 2:
                c_fail = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_fail) == PASS:
                c_pass = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_pass) == FAIL:
                c_fail = next_c_pass; n = 2; offset = 0; break
            elif test(next_c_fail) == FAIL:
                c_fail = next_c_fail; n = max(n - 1, 2); offset = i; break
            elif test(next_c_pass) == PASS:
                c_pass = next_c_pass; n = max(n - 1, 2); offset = i; break
            else:
                j = j + 1
        if j >= n:
            if n >= len(delta):
                return (delta, c_pass, c_fail)
            else:
                n = min(len(delta), n * 2)
```

@AndreasZeller

\302\n

@AndreasZeller

And if you have, say 2,000 lines of nroff input, it will reduce these to just two characters

---

first talk on delta debugging

@AndreasZeller

So, this is what I presented in Toulouse in 1999, and it was very well received, big applause and all. But after the talk, right as I get out of the room, there's a senior professor from France who is very agitated. He shouts at me (with French accent)

“I would never have thought

@AndreasZeller

"I would never hav sought

that something *so simple*

@AndreasZeller

zat somesing so simple

---

could be accepted at a scientific conference”

@AndreasZeller

could be accepted at a scientific conférence!"

@AndreasZeller

Yeah. Here we were. How could I continue this story?

Intellectual superiority

@AndreasZeller

Maybe on how the scientific styles differ from country to country. If I wanted to impress my audience with my intellectual prowess, filling the talk with formulas and special terms such that nobody can follow and everyone recognizes my superiority (I hear they do this in France), well, then delta debugging would not be it.

Impostor syndrome

@AndreasZeller

I could also talk on how this raised doubts in me on whether I'd done the right thing. Anybody could have come up with this! How did I deserve to be called a scientist? And how does the audience not see I am a fraud?

Impostor syndrome

@AndreasZeller

I could also talk on how this raised doubts in me on whether I'd done the right thing. Anybody could have come up with this! How did I deserve to be called a scientist? And how does the audience not see I am a fraud?

Simplicity

@AndreasZeller

However, the way I'd like to spin the story here is simplicity.

---

something so simple

@AndreasZeller

Remember: "somesing so simple".

---

complexity

@AndreasZeller

What's the alternative to simplicity? Well, complexity. And complexity

complexity is our enemy

@AndreasZeller

Is our enemy.

---

control complexity

@AndreasZeller

which we have to control

The purpose of software engineering
is to control complexity, not to create it.”

– Pamela Zave

@AndreasZeller

As put forward in this wonderful quote by Pamela Zave

Making complex things simpler

@AndreasZeller

So let this be the essence of SE. and, by the way, of delta debugging.

---

one year

@AndreasZeller

And I'd like to point out that it had taken me *one year* to make delta debugging as simple as it was

---

simple = hard

@AndreasZeller

So making things simple is hard work

simple = better

@AndreasZeller

But simplicity makes all of our lives much better

Debugging Reinvented

@AndreasZeller

And by the way, while praising simplicity, I'd like to take the opportunity to honor Andy Ko and Brad Myers, whose approach to debugging is for me the epitome of simplicity.

---

simple = better

@AndreasZeller

But then, such simplicity is hard to find.

graduate school

@AndreasZeller

A few years ago, I visited a high-profile graduate school. One of the best universities in the country, extremely selective, extremely ambitious. So there's 20, 25 students in the room, and they tell me they are expected

---

one paper per year

@AndreasZeller

to publish *one paper per year*. But not anywhere,

---

one paper per year
at a flagship conference

@AndreasZeller

but not anywhere – at ICSE, FSE, ASE.

one year making things simple

@AndreasZeller

Now remember: I spent one year refining delta debugging. I don't think I had a paper in 97 or 98.

---

getting a paper accepted is easy

@AndreasZeller

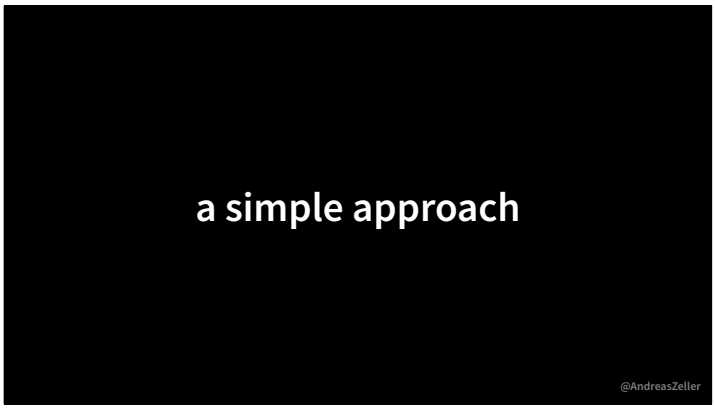
But then, fortunately, it turns out that getting a paper accepted is easy. All you need

---

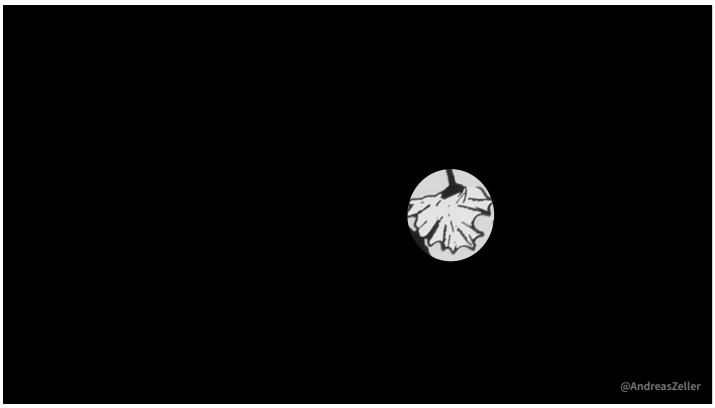
a recipe

@AndreasZeller

is a recipe – for doing research that will get accepted. One such recipe is

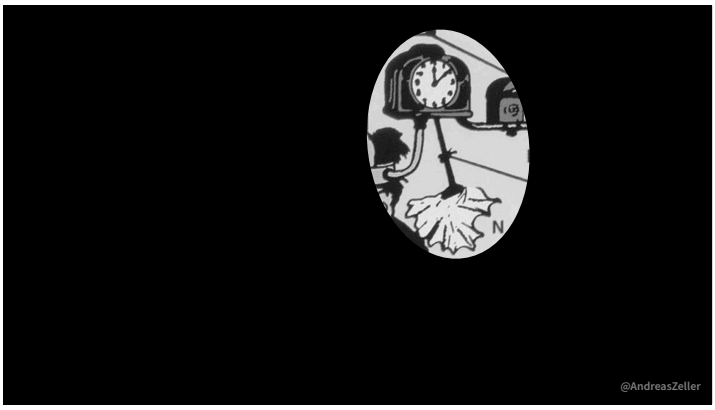


to take a simple approach

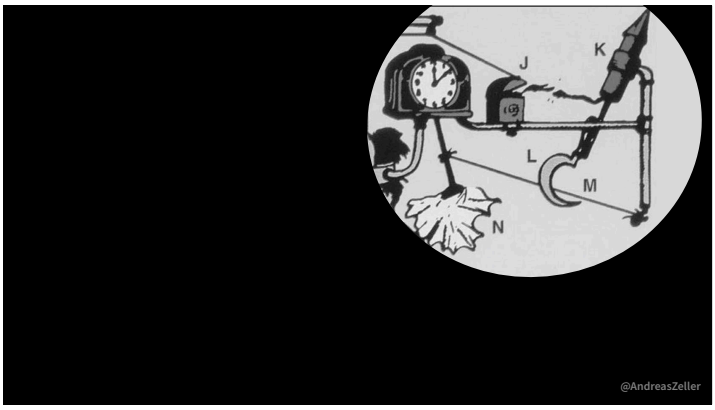


say, something we use every day. A napkin, for instance.

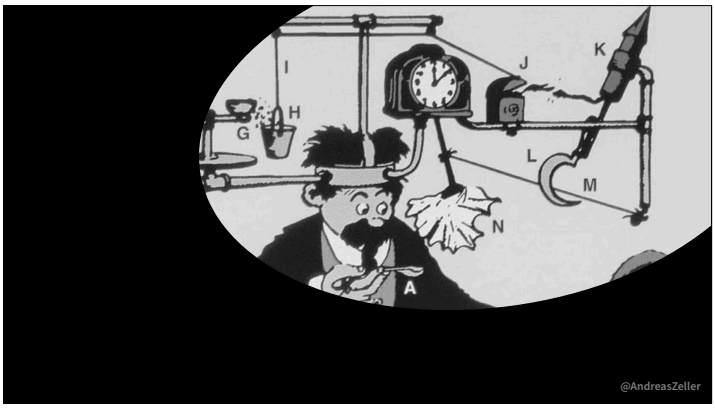
Picture source: Wikipedia



You then add some increment to it. Make it more automated. Say, add machine learning.

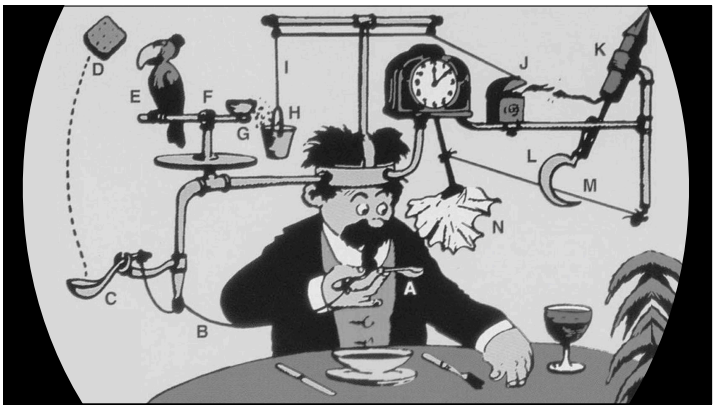


Make it dependent on context, such that it will work well in that context. If not, cut it off.



Integrate all this into the user's environment. Just continue adding and adding

---



Until it gets better. Say, 2% more precision. 5% more area under the curve. Errors found.

Reviewers get this

@AndreasZeller

This is so great, because even if reviewers do not understand your approach at all, they will understand the improvement.

---

a complex approach that is better

@AndreasZeller

What you then have is a complex approach that is better

~~complex is better than simple~~

@AndreasZeller

But then, is this really the case? With such complexity, who wants to re-implement your approach? Who wants to *use* it?

The purpose of software engineering
is to control complexity, not to create it.”

– Pamela Zave

@AndreasZeller

Maybe it is time to apply our principles to our own research.

---

more recipes

@AndreasZeller

There's more such recipes, of course; and you may argue: So what? Who cares about a paper too complex getting in? Well, the problem is that such papers

---

obstruct scientific progress

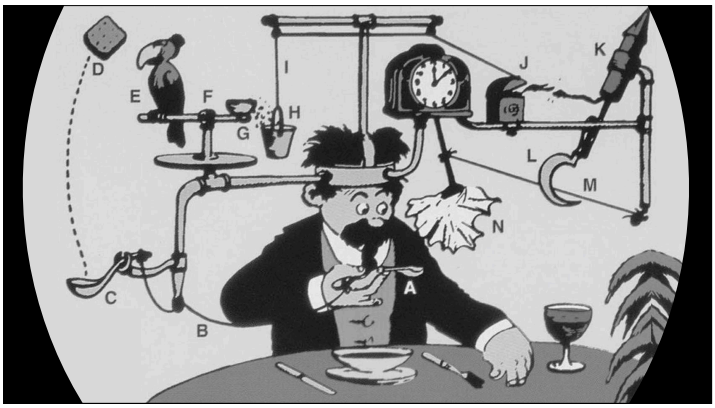
@AndreasZeller

obstruct the scientific progress – because the only way to get even better results

more complex = even better

@AndreasZeller

is to build something that is even more complex. This helps nobody.



So we end up with contraptions that are more and more complex, and yes! They automate something! And yes! They are better than manual work! Yet, this helps nobody.

Picture source: Wikipedia

Would something so simple

@AndreasZeller

And I wonder: Would something so simple

as delta debugging

as delta debugging

@AndreasZeller

---

get accepted at this scientific conference – *today*?

get accepted today?

@AndreasZeller

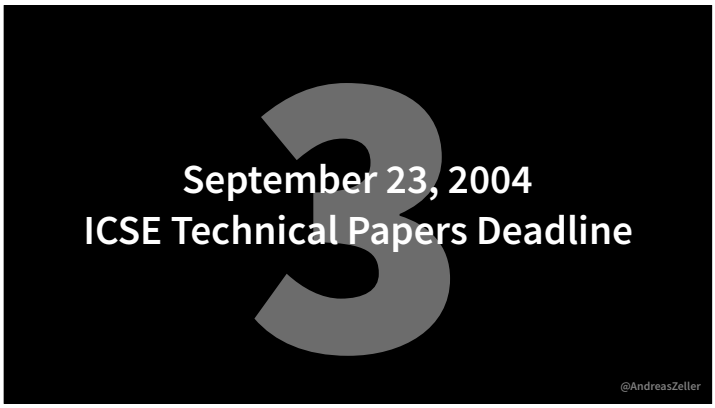
Okay, we're short on time, so let me close with the third story.

123

Three Stories

@AndreasZeller

Again, five years later. It is a Saturday morning in 2004,



and it is the day of the ICSE deadline. You know ICSE deadlines, right?

I'm a tenured professor,

full professor

@AndreasZeller

actually full professor

---

delta debugging + DDD

@AndreasZeller

a position which I got through delta debugging and DDD

---

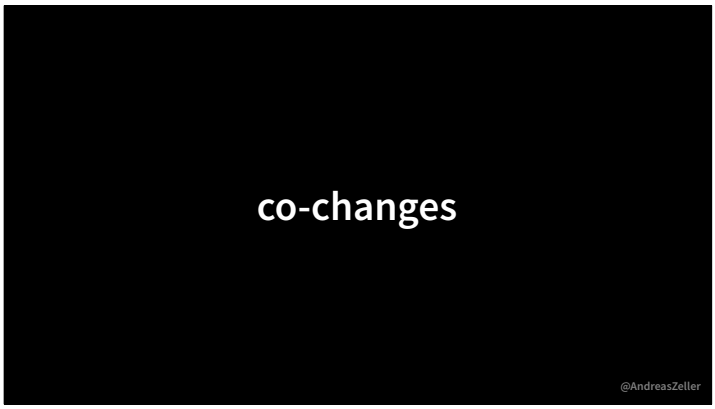
students

@AndreasZeller

a new thing: I now have students.

Here's one. You know that guy? It's Tom Zimmermann.

With Tom, we systematically analyze version histories



Specifically, we look for co-changes,

People who changed A
also changed B

@AndreasZeller

That is, changes involving multiple components at once.

---

src/file.c ⟺ doc/schema.jpg

@AndreasZeller

We find relationships. For instance, whenever someone changes file.c, the file schema.jpg is also changed. Why is that? Turns out file.c has an embedded SQL statement, and schema.jpg is a picture of the database schema. When the schema changes, so does the SQL statement. Find *that,* static analysis!

---

recommend changes

@AndreasZeller

We can go and recommend changes

src/file.c ⇒ doc/schema.jpg

@AndreasZeller

Change A, you also need to change B

---

precision and recall

@AndreasZeller

And we struggle with accuracy metrics such as precision and recall, which are all new to us. (As also for the SE community)

September 23, 2004
ICSE Technical Papers Deadline

@AndreasZeller

So it is the Saturday of the deadline; deadline is around noon in Europe;

precision and recall

@AndreasZeller

And we are still struggling with these metrics. This is when Tom calls in at 10am – two hours before the deadline.

---

precision and recall > 90%

@AndreasZeller

He says he has found a way to boost precision and recall above 90%. And I tell him, this is great, but this sounds to good to be true, so please check and re-check.

---

training from the testing set

@AndreasZeller

One hour later, one hour before the deadline, he finds he has accidentally trained from the testing set. So, we're back to our old values, and we submit.

reviewers are unsure

@AndreasZeller

The reviews are mixed. The reviewers clearly don't know what to do with this

---

but accept anyway

@AndreasZeller

but accept anyway

Mining version histories
to guide software changes

@AndreasZeller

The paper title is "Mining version histories to guide software changes"

1,200+ citations

@AndreasZeller

Today, it has more than 1200 citations

1,200+ citations
ICSE *n*-10 most influential paper award

@AndreasZeller

Three years ago, Tom, Stefan, Peter, and I got the most influential paper award.

@AndreasZeller

So again, how do I spin this story? I could tell something about

Quality assurance in research

@AndreasZeller

how important it is to do thorough quality assurance, how to ensure your results are reproducible and all, and yes, it is.

We were *so* lucky

@AndreasZeller

I could also spin how lucky we were, as Gail Murphy and her student Annie Ying were working on exactly the same topic, with the same results, but decided not to go for ICSE because they wanted better precision and recall. Luck is the most important factor for success.

Innovation

@AndreasZeller

But the lesson to be learned from this, for me, is innovation. Actually, our concerns about

numbers

@AndreasZeller

precision and recall

---

did not matter

@AndreasZeller

did not matter.

---

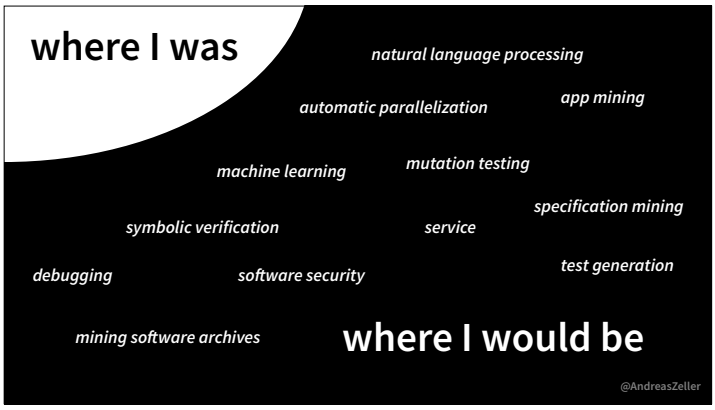
nothing to compare against

@AndreasZeller

simply because there was nothing to compare against.

Our approach was entirely new, finding things that no-one else did.

It was new, and new was better.



Going from debugging to mining software archives was one step towards something new, and I have kept on moving since then, exploring dozens of new fields – sometimes successful, sometimes not so – but always learning, always progressing.

with many great students

@AndreasZeller

And that's not me. That's me and many great students, whom I admire and love very much.

---

work that is simple and relevant

@AndreasZeller

And work that would be simple *and* have impact in practice

Delta debugging

Mining software archives

work that is simple and relevant

Fuzzing with code fragments

Checking app behavior against app descriptions

@AndreasZeller

* Delta debugging narrows down failure causes
* Mining software archives yields empirical findings
* Grammar-based fuzzing tests JavaScript interpreters in all browsers
* Apps are checked against descriptions and categories (at Google/Microsoft)

we need patience

@AndreasZeller

But we'd also need patience

---

entering a new area = 1–2 years

@AndreasZeller

Because if you enter a new area, it takes a year at least to understand how it works

---

getting cited = many years

@AndreasZeller

And if you have something really new, it can take many years until it gets cited.

a trusting environment

@AndreasZeller

I was very glad I had an environment that would trust me:

Saarland Informatics Campus

@AndreasZeller

the Saarland Informatics Campus in Saarbrücken, Germany

---

the hire with lowest # of papers

@AndreasZeller

When I got hired, I was the one candidate with the lowest number of papers.

never evaluated my research

@AndreasZeller

Nobody ever checked my publication counts.

---

impact alone counts

@AndreasZeller

The only thing that matters, they told me, will be your impact

---

even if it takes decades

@AndreasZeller

even if it takes years or decades to build

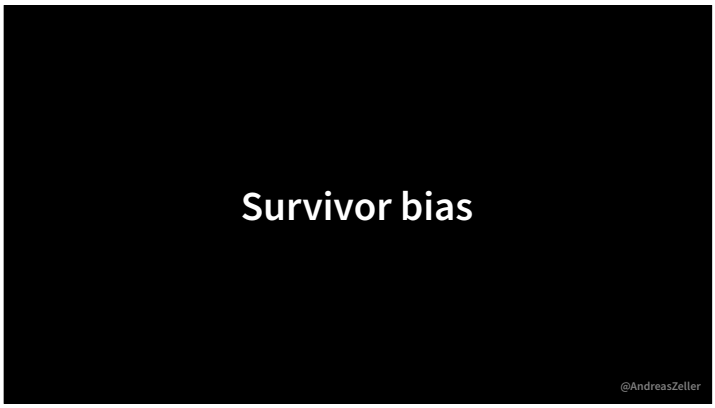
They took enormous risks, they put in an enormous trust. They trusted me all this time –

and here I am today. Thank you so much.

(short pause)

In this moment, as I am standing here, I realize how lucky I was, again and again. Most of us have to struggle hard in our daily work, trying to fulfill the most absurd incentives and regulations.

[Source: https://en.wikipedia.org/wiki/Sisyphus]



I missed almost all of this. I was lucky, and my luck is why I am standing here.

But whether you are lucky or not – these hold for all of us:
If you are in the light, go explore the dark

search for *relevant* problems

@AndreasZeller

Find out what is relevant

---

search for *simple* solutions

@AndreasZeller

Find out simple solutions

---

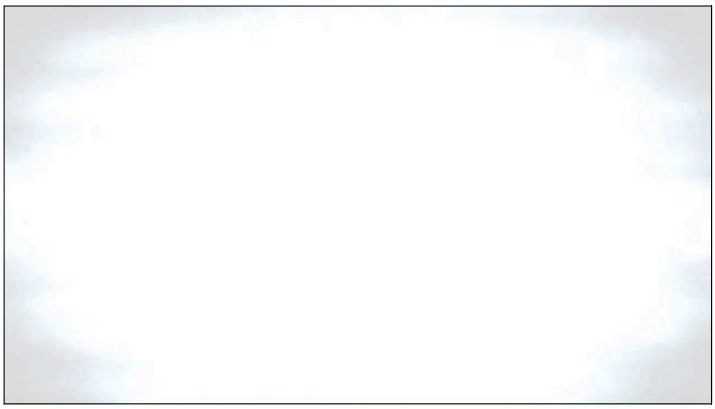
keep on innovating

@AndreasZeller

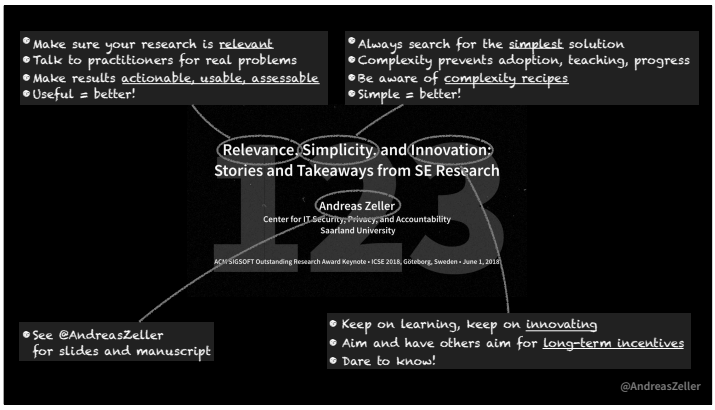
Keep on innovating

As the Romans say, "sapere aude": Dare to think for yourself, dare to be wise;

Or simply: Dare to know.



That's it folks – three stories, three takeaways



on • relevance, • simplicity, and • innovation.

Now **go out** and create masterpieces of Software Engineering –

and • see you next year in Montreal!