Elephant in the Room

Who writes the bad code?

Hi, my name is Niclas Hedhman, a long-term Apache Software Foundation contributor. I have worked 32 years as a software developer, and over the years I have made many observations. And latest observation is about something that is so obvious that we don't notice it.
The "Elephant in the Room" is an expression in English, about the obvious things that no one dares to talk about.
In this presentation, I want to bring up the topic in the Software Industry, that no one dares to talk about, but is paramount to bring us to the next level.

Observations

- Who has seen beautiful code?

  -- Almost Everyone

- Who works with beautiful code?

  -- Very Few

- Who creates new beautiful code?

  -- Even Fewer

I mentioned that I have made observations. We all make observations.
Almost everyone has seen beautiful code. It is concise, it is simple yet powerful, it speaks to us in an easy to understand abstraction, it is easy to use, easy to extend. We can quickly do our job, get things done.
However, if we ask people if the work on beautiful code, most people say No. So where is this code coming from? Well, very few developers are responsible for these gems.

Observations

- Who has seen bad code?

  -- Everyone

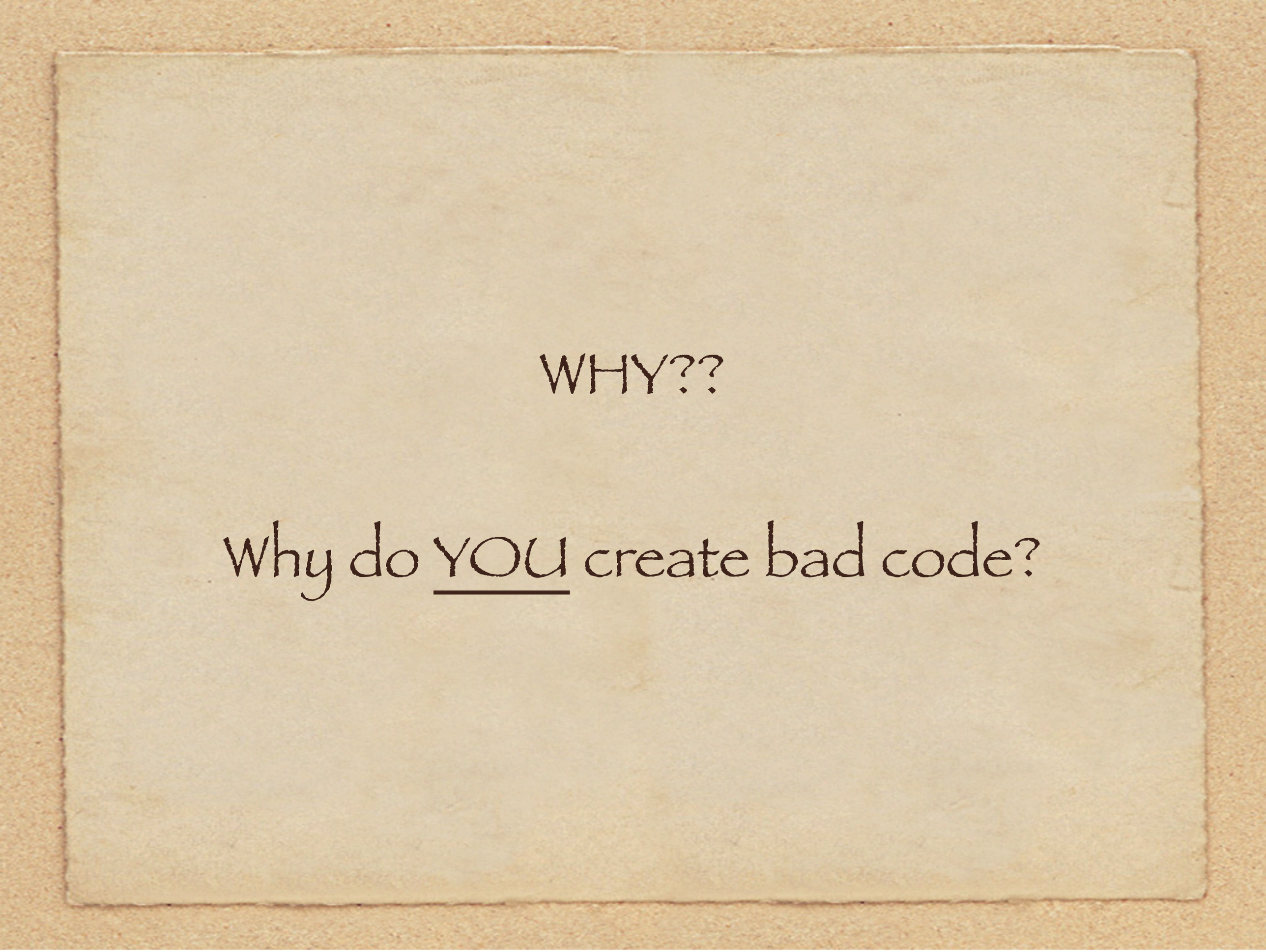
- Who works with bad code?

  -- Almost Everyone

- Who creates new bad code?

  -- Everyone

On the other hand, Bad Code is everywhere. We have all seen bad code. Many of us are fighting bad code on a daily basis, and we are all guilty of creating more bad code.



But WHY? Why do you create bad code? You know it is bad when you write it. Why, why, why?

Why write bad code?

- "It is already bad. I can't improve it..."
  - "No tests..."
  - "Too complex..."
  - "Too risky..."
- "I need to deliver. I will fix it later."
- "It is not THAT bad..."

If we listen to ourselves, we will hear statements like these;
It is already bad. I can't improve it, because there are no Tests or that the codebase is too complex. It is too risky to make needed changes.
I am so busy now. We have a deadline, and we will fix this later when we have more time. Right now, we need to focus to push this out the door.
Hey, I don't think it is bad. My teammates are unhappy about it, but I think they are not smart enough to appreciate this.

I say "Buuuuull.....!"

Those are EXCUSES...

And I say that all of such statements are buuuuuulllshit. Those are excuses, because otherwise we can't live with ourselves. How could we go to work, knowingly do a bad job and not be ashamed ourselves to the point of changing careers?

There is SOMETHING else.
SOMETHING much deeper.

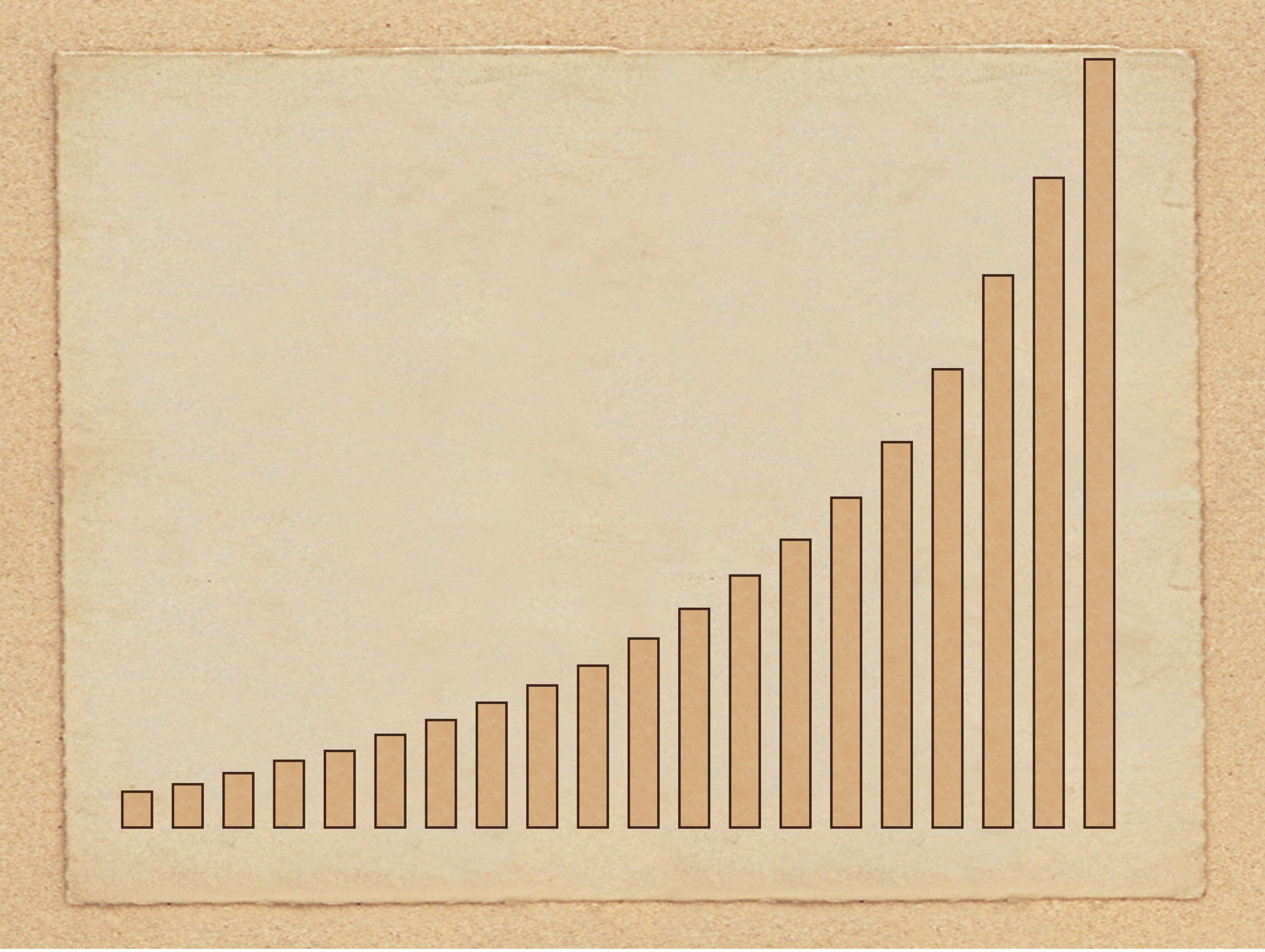
PROFOUNDLY DISTURBING!!

My hypothesis is that there is something much much deeper to the long-standing crisis in our industry. And I find it profoundly disturbing.

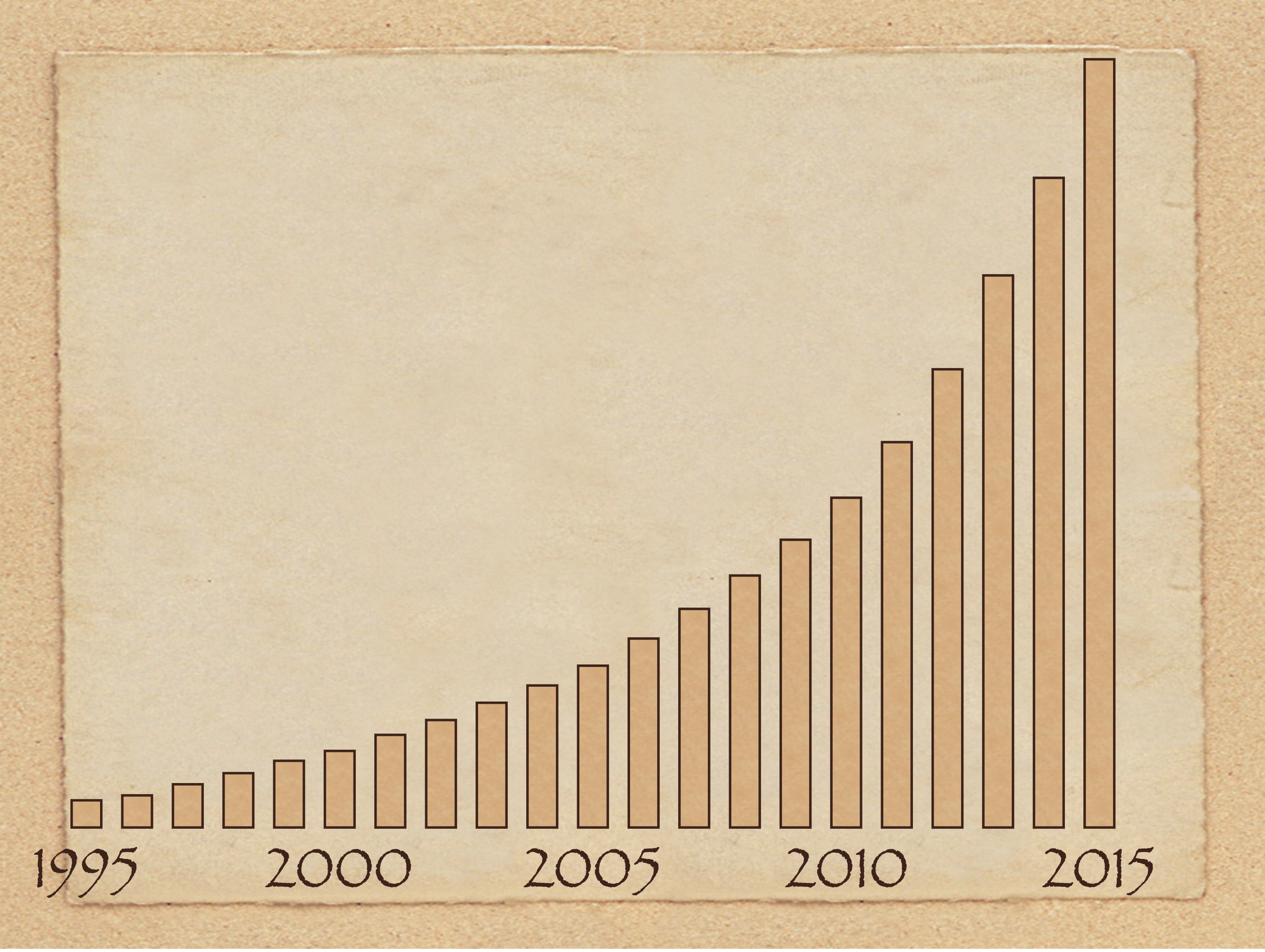
Our Industry

- Constantly Innovating
- Almost No Limits
- Exponentially MORE TO LEARN
  - Doubles every 3 years??
- Open Source
  - Standing on the Shoulders of Giants

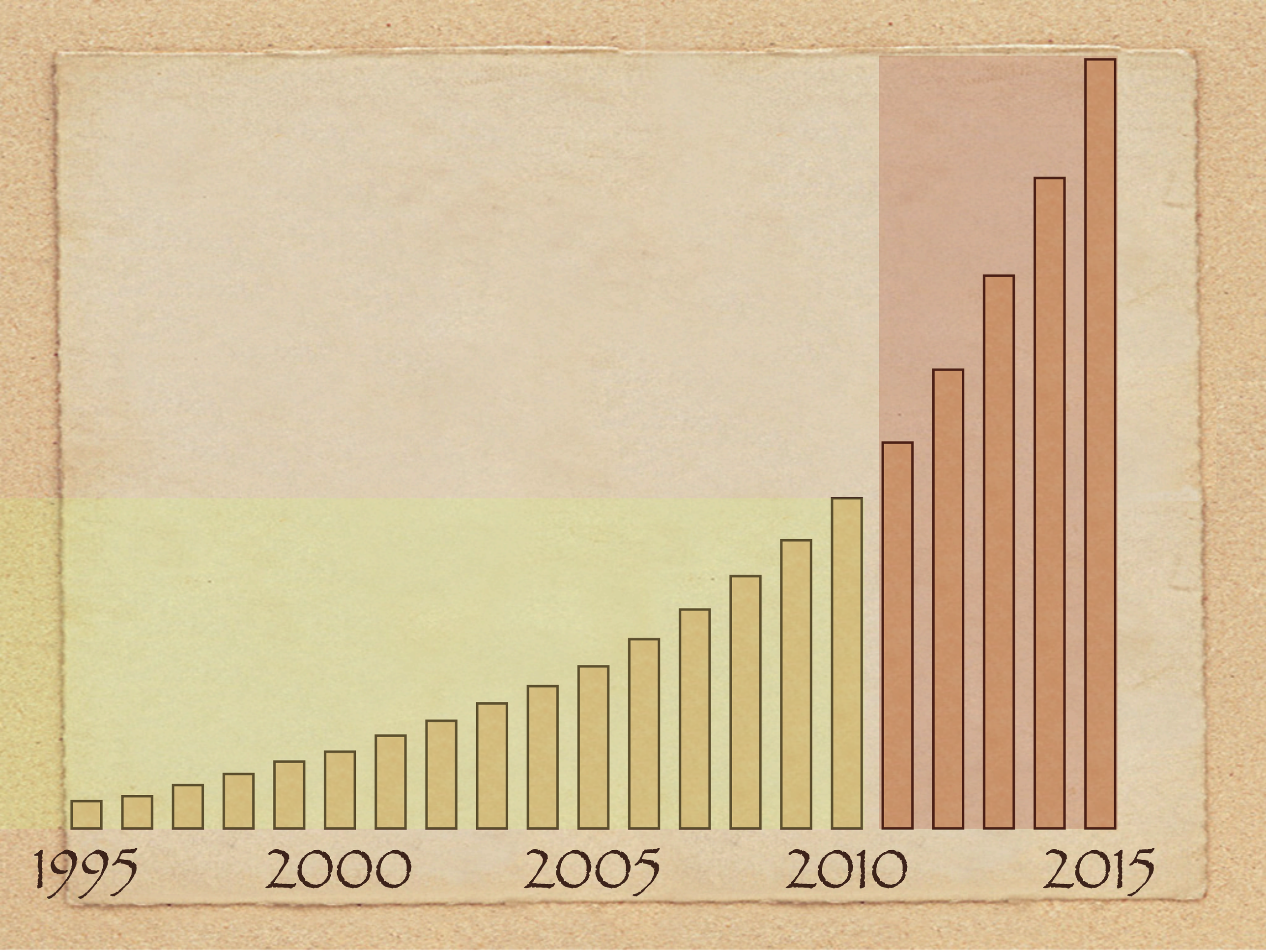
First, let's look at some facts.
Our industry is changing at a remarkable speed. New technologies appear almost weekly, and have done so for the last 30 years that I have been around. There are practically no limits to what people create. And the amount of things there is to learn is staggering.
Open source is now so common place, that we never ever start from scratch. We build upon existing platforms, combining existing things in new ways, add our own innovations on top, and quite often we release that as open source for other to build on top of that, or be inspired to build something else. This result in an unprecedented acceleration of output.



Can anyone tell me what this graph is?
<want an answer from audience>



This is a the growth curve of number of programmers in the world. The exact numbers are not important, although it is estimated to be in the 50-100 million range.
What is important is that it doubles every 5 years or so. It has been doing that almost non-stop since 1950. Every 5 years, the number of programmers in the world has doubled.



What do we know about exponential growth? Well, 2 things. The good news is that it can’t go on forever. Soon there is not enough population on the planet. The bad news is that half of all programmers have less than 5 years of experience. In reality, it is even a lot worse than that, since many leave the profession within 10-15 years.

Our Industry

- Young & Inexperienced
  - Doubling every 5 years == 50% less than 5 years experience
  - Not enough knowledge transfer
    - Don't understand enough
    - Same mistakes over and over again

So, we have an industry that is extremely inexperienced. There is a perpetual lack of experience. Worse yet, since the technology landscape changes so quickly, there is not enough time to become experienced before the technology is out of fashion and every cool kid on the block have moved on.
This leads to the same mistakes are made over and over again.

Our Industry

- Top-Down Edicts
  - SQL RDBMS
  - OLE, COM, DCOM, LINQ
  - EJB, Java EE, JSP, JSF, frameworks
  - Enterprise Service Bus
  - Big Data, Hadoop, Spark
  - Microservices

Most programmers don't invent new technologies for others to use. We adopt technologies that got talked about a lot. We trust others to have figured out what is good for us. We don't apply critical thinking. Let me repeat that. We don't think critically, and in faith-like fashion believe what we are told. That is a clue to where I am going with this.

MYTHS!!!

Let’s move on to myths in our industry. And there are a lot. Since we act like a religious sect, many myths have risen incidentally or on purpose.

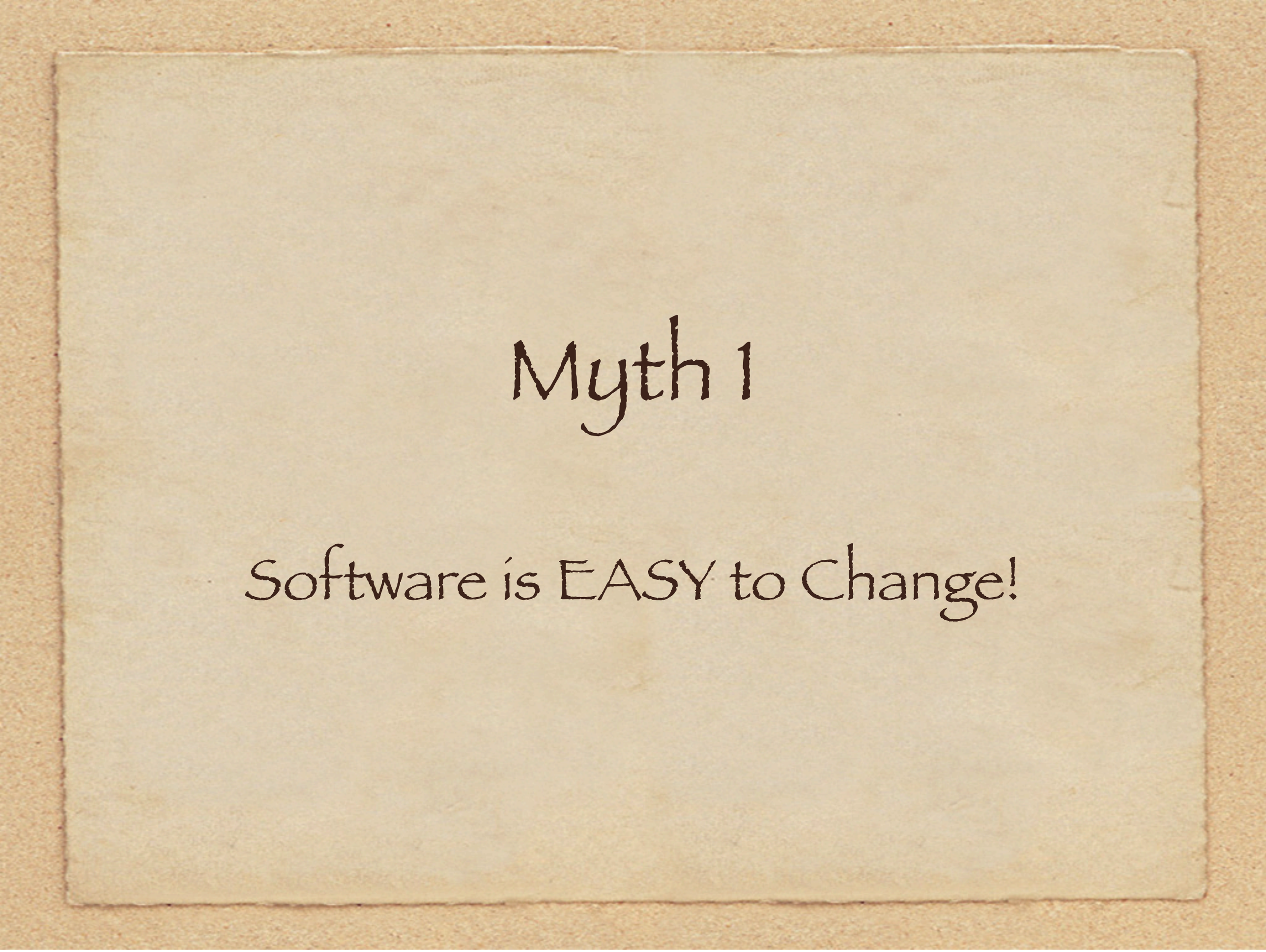
Our Industry

- Myth Propagators
  - Vendors - Sell products/services
  - Speakers/Authors - Sell books/services
  - Bloggers - Sell services
  - Colleagues - Sell Prestige/Politics

Myths are created and spread by certain types of people. People that we respect and therefor we are very vulnerable to what they have to say.
Vendors try to sell us product and services.
Likewise speakers at conferences, authors of books and bloggers that we read, more often than not are also selling their books, sites or services.
At work, we listen to people more senior than us, typically those colleagues that have a string of successes behind them, but if it is a big company there are politics and prestige at play, which we should also question.

So what are the MYTHs?

So let’s look at typical myths that are so called common knowledge.



Myth number 1. Software is Easy to Change.
We have grown up with the notion that software is so ..... well.... soft. With a little bit of typing, we can change it to do something completely different. Redesigning electronics takes days, software takes minutes.

Reality Strikes Back

Almost All Software is HARD to Change!

But hey... Reality is a nasty beast. It doesn't listen to myths and just comes to bite us over and over.
Most software is not only incredibly hard to change, it tends to never die either. Once written and deployed somewhere, it takes forever to get rid of it, no matter how trivial or useless it is.

Myth 2

More People = More Result

This myth is prevalent in company management. If we need to do twice as much, we hire twice as many programmers. That will do it.

Reality Strikes Back

The Mythical Man-Month
More People = Less Productivity

The Mythical Man-Month was a book published in 1975. 1975. 40 years ago. The author Fred Brooks observed that adding more people to a project behind schedule would in fact make the project even slower. Eventually, the communication required to coordinate between people will take more than all the time of each person.

Myth 3

Programmers are INTERCHANGEABLE!

Another favorite among managers in large companies, is that programmers are interchangeable parts. If one programmers leaves, we just pick a new one from the street to replace her.

Reality Strikes Back

Knowledge of Software is in the BRAIN!

Except that doesn't work. Software knowledge is not in the code. It sits in the brain of the people that created the code. If you have ever come in to rescue a codebase where all the previous programmers left without any handover, then you know what I am talking about. If the creator of the codebase quit, it takes 2 new people to replace her, and it may take them a year before they think they understand what the creator meant, perhaps never.

Myth 4

Abstractions are GOOD!

Here is one of my favorites. We wouldn't have the Internet and all its fascinating sites on it, if it wasn't for abstractions. If we had to write binary instructions for the CPU, we would have problem to create the simplest of programs. Programming languages, protocols, data formats, frameworks and much more are abstractions helping us to do our job.

Reality Strikes Back

Most Abstractions CONFUSE!

(because bad abstractions)

But most abstractions are in done in the programs that we write. We create our own small clever abstractions, without enough clarity, without the experience of what makes good abstractions, and we make a mess. Our own abstractions, more often than not, are horrible, confusing and outright detrimental to our projects.

Myth 5

Methodology X solves Y

There are many peddlers of methodologies. It was Objectory with use-cases in the late 1980s, then came the Rational Unified Process and many other so called CASE (Computer Aided Software Engineering) tools, settling on Unified Modeling Language, or UML. Kent Beck introduced us to Extreme Programming, which was the first agile methodology as a revolt against the so called waterfall methodologies. Scrum, Kanban and others have been touted in the last 10 years or so. All promising to solve the complexity of creating software.

Reality Strikes Back

Success STILL Depends on HEROES!

My observation is that no methodology works as advertised. Practically all software projects depend on heroes. People that make it work, no matter what the methodology is. For new projects, the heroes are skilled at getting things started. For codebases that are in maintenance or evolutionary mode, the heroes are the folks that don't mind the bad code that they work with.

Our Industry

- Geniuses
- Smart Developers
- Average Developers
- Weak Developers
- Bad Developers

Back to our industry. I would say that we can categorize the people in our industry in the following manner.
Geniuses, Smart, Average, Weak and Bad Developers. Let's take a look at the characteristics of each.

Geniuses

- Write Simple Libraries, that does a lot
  - Never done, Evolves
  - Maintained, Improving
  - Enough Tests
  - Humble of their contribution
  - Rewrites without Effort, *yet* Compatible

Geniuses write very simple, re-usable libraries that does a lot. They stick around and keep evolving that library with small and regular improvement and new features. They write enough tests, relevant tests. They are often very humble, knowledgable that they are not perfect. They can often rewrite the entire library from scratch, in a better way without breaking compatibility.

Smart Developers

- Write Complex Libraries, that does a lot.
  - Code 80% done, undocumented
  - Not maintained, stale a year later
  - Deteriorate badly, can't be replaced
  - Too much Tests
- Proud of their achievement
- Want Rewrites, Can't Succeed

Smart developers on the other hand, write complex libraries. Often they leave a lot unfinished, and since they don't stay for very long, moved on to new exciting things, the library goes stale rather quickly, no one understands how it works, and the tests are too many and too coupled to the code.
Smart developers are rather proud of themselves and their work, but they are not able to rewrite the code that they created, because of poor testing strategy and inexperience.

Average Developers

- Don't Write Libraries
  - Write Production Code
  - Hack libraries, Hack Everything
  - Monkey patches
    - "Tests Stopped working.", "Disable!"
- Curses the Smart Developers
- "Rewrite? What is that? We Deliver!"

Average developers are normally heroes in software maintenance in large companies. They hack together anything to get the bug fixed or the new feature out the door. And if tests breaks, they simply disable the tests, because they need to ship code. They are also upset with the Smart developers that left behind dependencies on abstractions and libraries that are hard to use, full of bugs and completely undocumented.

Weak Developers

- Don't Care about Code
  - Don't know if code is good or bad
  - Don't think twice about what they write
  - Don't Listen to Advice
  - Don't Learn from mistakes
- A Rewrite Always Needed Afterwards

Weak developers don't care about anything. They go to work. Write some code. Go home. Repeat. They never learn, they don't listen and they couldn't tell good code from bad even if it was shown to them.They should not be programmers, and they seldom last very long.

Bad Developers

- Want to Write Code
  - Not Allowed to, without Supervision
  - Smaller Tasks, Easily Correctable
- No Damage
- Transient - will graduate

Bad Developers are not as bad as Weak developers, because both they and the people around them realize that they are inexperienced and not very good. So they want to learn, they are eager and often they will transition into being better programmers. But some, will only become Weak developers. It is a transient phase, and nothing to worry about.

Myth 6 - Perception

- Geniuses ~20%
- Smart Developers ~40%
- Average Developers ~30%
- Weak Developers ~0%
- Bad Developers ~10%

That brings us to the next myth. If we ask people which category they belong to, but without the exact definition of those categories, then we will get a result something like this.
60% in the Genius and Smart categories. and only 30% or so think that they are average.

Reality Strikes Back

- Geniuses - <1%
- Smart Developers -10%
- Average Developers -50%
- Weak Developers -30%
- Bad Developers -10%

But in my experience, the numbers are something like this. Very few geniuses, a handful of smart developers and the bulk of people in the average and weak category.
Think about that for a moment.

Myth 7

Primus Inter Paris

(greek; First among Equals)

"I Can Do That."

So, when the Genius Developer shows us his remarkable, simple masterpiece, beautiful design, elegant abstraction, ease of use.... We think "I Can Do That"
In Greek it is "Primus Inter Paris", the First Among Equals, the view that truly outstanding people are still just one of us, no different, we can all do it.

Reality Strikes Back

Illusory Superiority

&

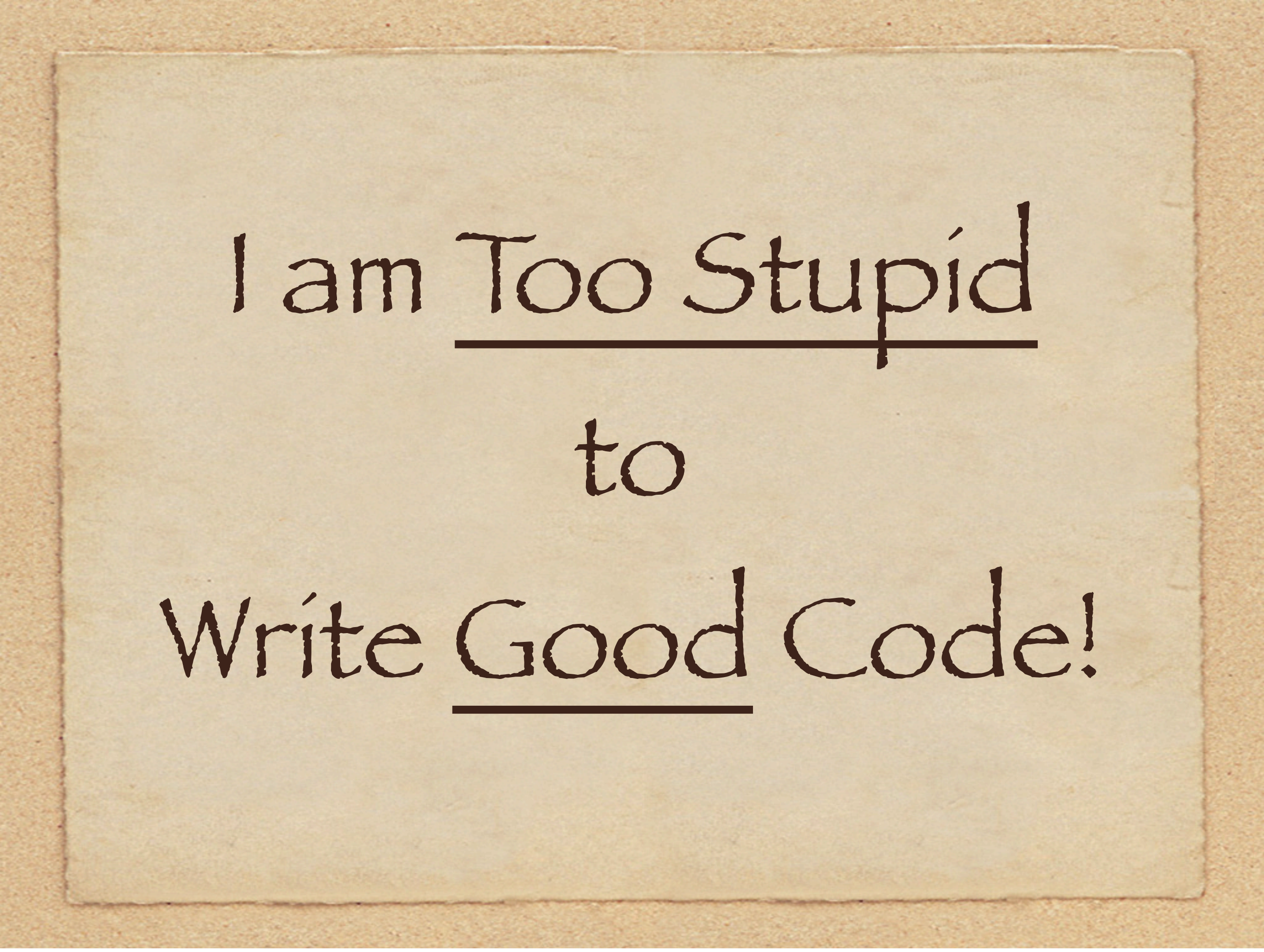
Dunning-Kruger Effect

When we see the end result, everything seems obvious. But the road there is really obscure and we ordinary mortals are simply too stupid to realize our limitations. Just because we understand the end result, how it works, doesn't mean that we could invent it ourselves. Illusory Superiority complex, and its subclassification of Dunning-Kruger Effect, is essentially; We are too stupid to realize how stupid we are. We are confident in our abilities, we create complex software that barely works in the normal conditions, and fails miserably at edge cases.

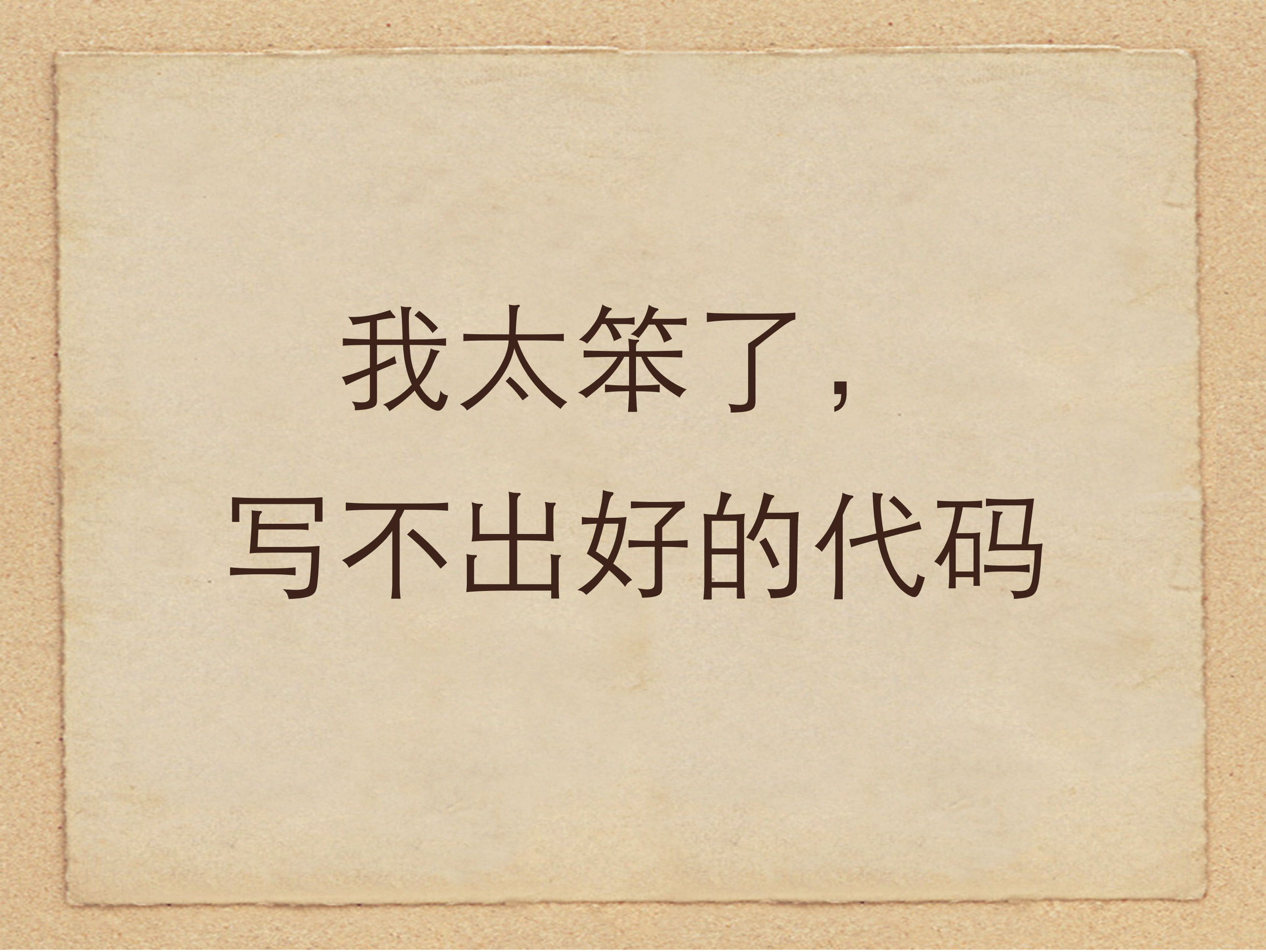
so...

Say After Me...

Say after me



I am too stupid to write good code.
Everyone now; Say it....



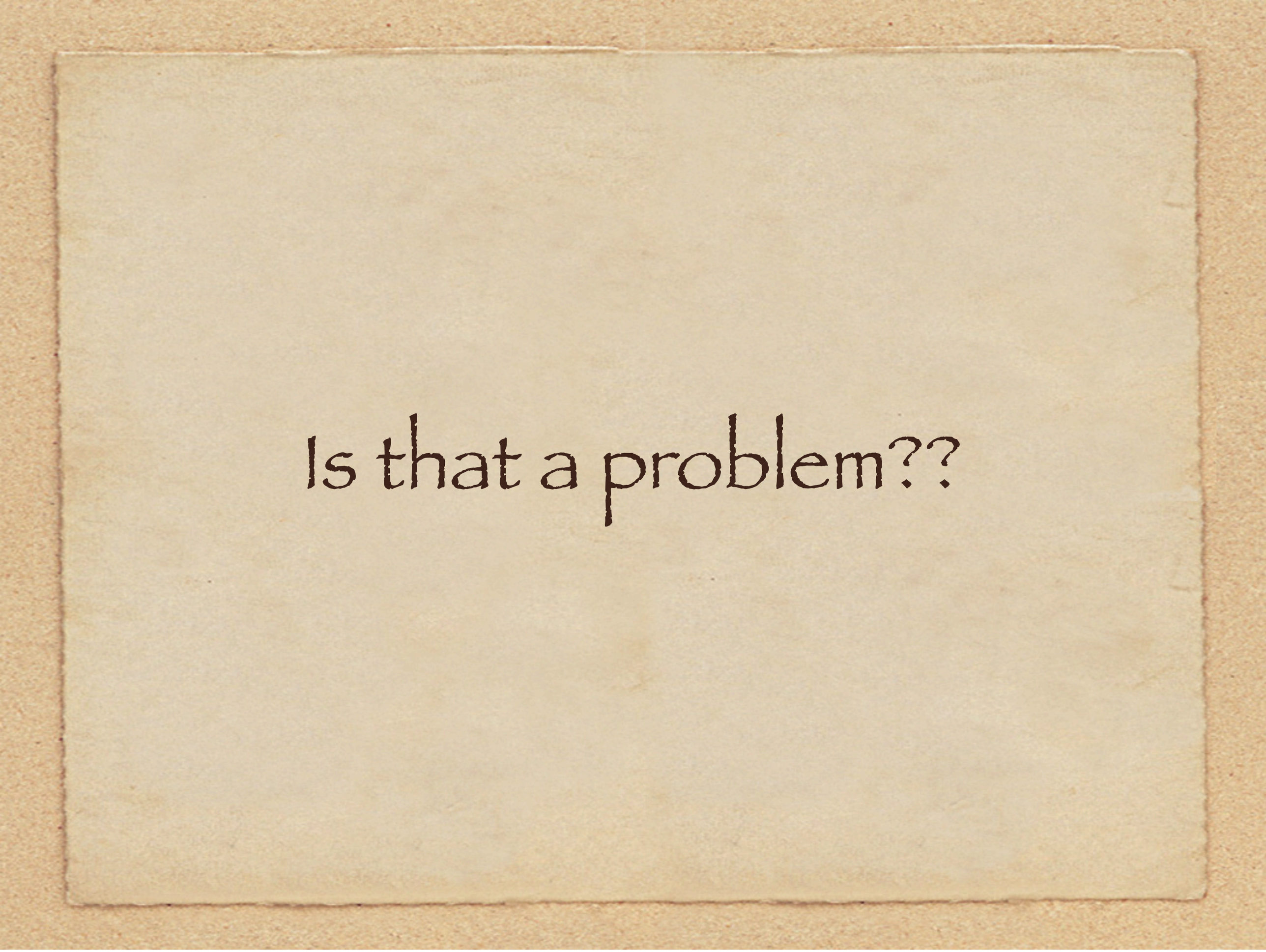
I am too stupid to write good code.
Everyone now; Say it....

THAT

is the

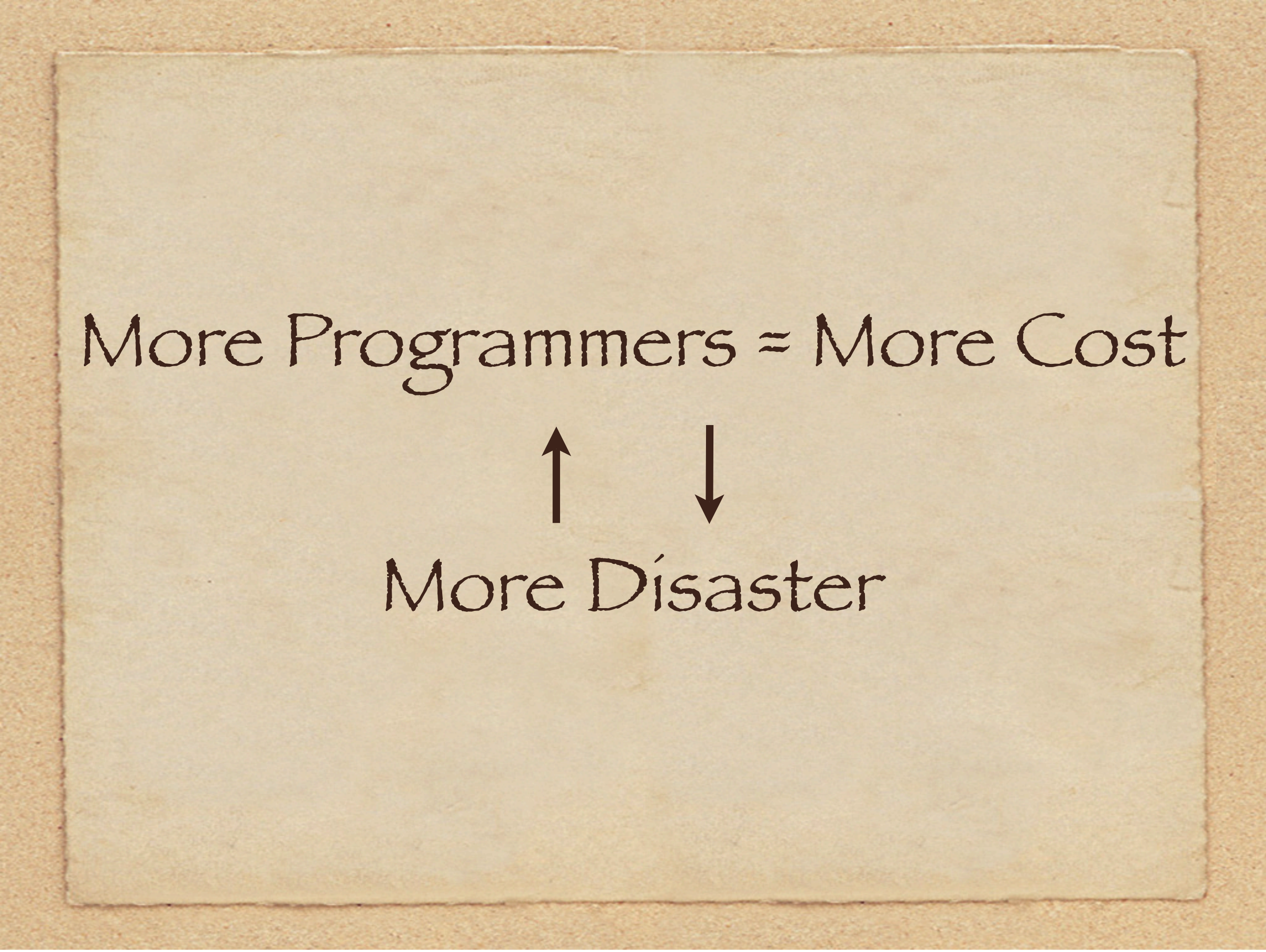
Elephant!!

And that is the Elephant in the Room.



Is that a problem?

Yes, firstly there are financial reasons.



More programmers are more expensive. And the more people involved in a codebase, the worse it gets.
Banks have armies of programmers, just to keep the programs alive. Changes are slow.

Employees

- Amazon = 269,000
- Google = 62,000
- Baidu = 41,500
- eBay = 34,500
- Alibaba = 26,000
- Tencent = 25,500
- Facebook = 12,000

If we look at so called Internet companies... What are all these people doing? Internet efficiencies was to reduce number of people needed.

Secondly, Social

Secondly, the current situation has social impact.

Unhappy programmers tend to change job. And I have tried to show that new people will most of the time make the situation worse. Another vicious circle, downward spiral into the big ball of mud.

And we should be satisfied with our jobs. We should feel joy when we go to work.

Another aspect of low retention is that many programmers will escape to other jobs, such as management, business analysts and even completely unrelated professions, because they are unhappy being programmers. And that leads to less experience, and more hopeless situations.

Solutions?

This presentation has been very negative, maybe even offensive to many people.
But are there any solutions?
Yes, I think that there will be, but it is far into the future. Some genius will come up with new paradigm for creating software. I expect it to be radically different from what we are doing now. The equivalent of introduction of the steam engine to the mining industry in the 18th century. Tools that removes our ability to create bad code, maybe artificial intelligence that has a complete different approach to software creation. Who knows?

Until then, I think we are stuck in the bronze age of computing, handcrafting every detail but without generational knowledge transfer, so we are not improving our collective skills.

I would like to leave you with this advice. The KISS principle - Keep It Simple, Stupid. It is something to keep in mind when you write code. Don’t make additional abstractions. Avoid generalizations. Avoid inheritance. Don’t write code that you think that you may need in the future. You are done, when there is no more code to remove.

Advice

Make It Run

Make It Right

Make It Fast

I would like to leave you with this advice. The KISS principle - Keep It Simple, Stupid. It is something to keep in mind when you write code. Don't make additional abstractions. Avoid generalizations. Avoid inheritance. Don't write code that you think that you may need in the future. You are done, when there is no more code to remove.

Advice

Don't Make it Worse!

Even if the code that you are working on is really messy and bad, please don't make the situation worse. Even if the codebase is horrible, when you make a change, improve it a little bit. If everyone improve a codebase a little bit every time you work on it, eventually it will improve quite a lot. And don't let your team mates make it worse either.

Questions?

niclas@hedhman.org

And with that, I would like you to thank you for listening.
I don't think we have time for questions, so if you do have questions just find me somewhere around here.