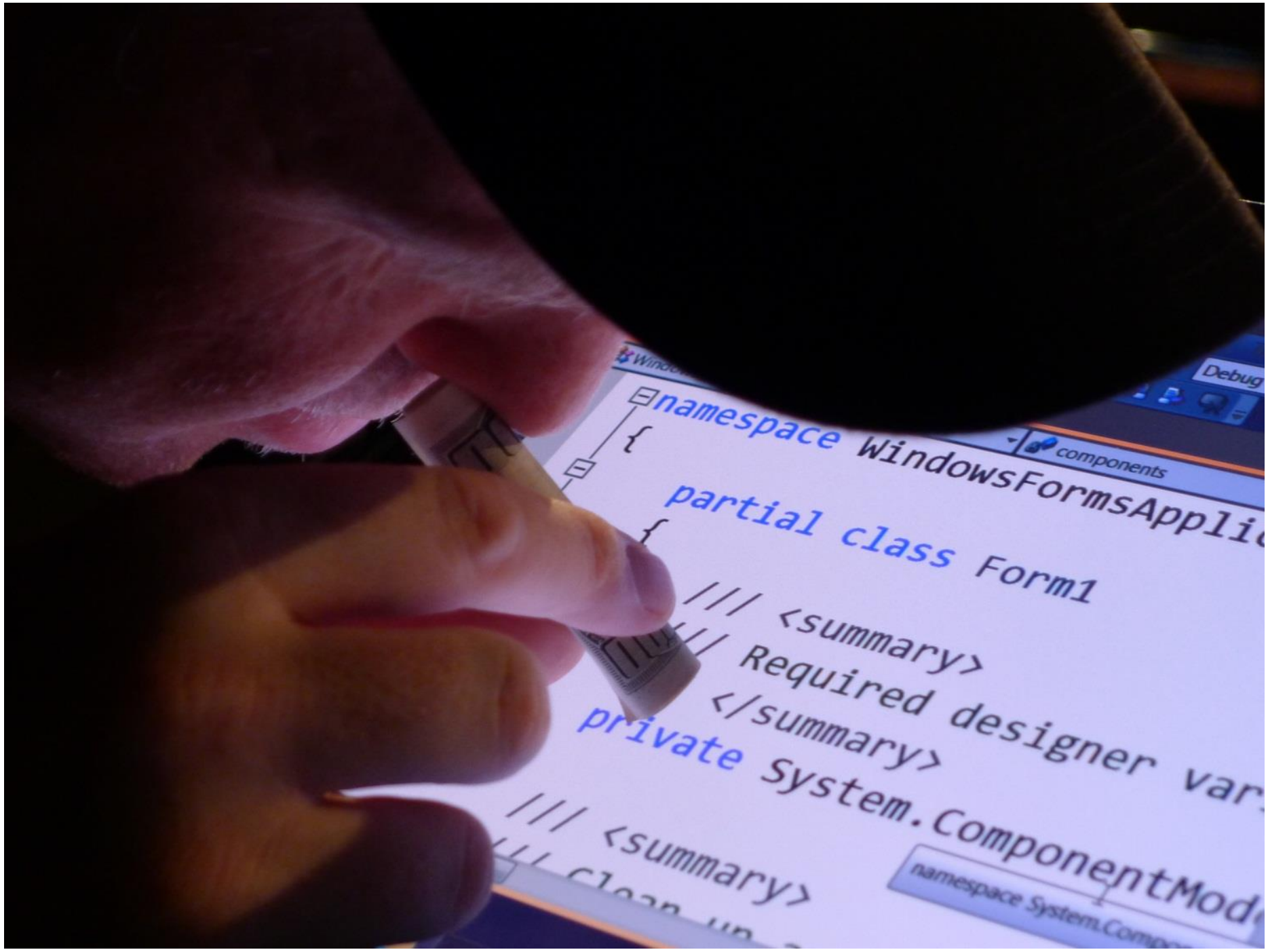
40 Years of Lessons from Software Development

Billy Hollis

Agent Provocateur

Too many developers are addicted to writing code. It's the lens through which they see every problem, and the main ingredient in every solution they devise.

- Some of you may have heard me talking about codeheads, who do lines to get their fix


namespace WindowsFormsApplic
{
partial class Form1
{
/// <summary>
/// Required designer var
/// </summary>
private System.ComponentMod
/// <summary>
Debug
components
namespace System.Compo

Code addiction drives many of our practices, and not always in a good way

- It excuses sloppy investigation of domain problems
- It suggests we have to throw something together to show to users, instead of understanding the problem and designing to it
- It means we often blow off things like testing

You don't have to do big design up front. But you need to do *some* design up front. And you really need some serious understanding up front. **You can't solve a problem you don't understand.**

Soft skills are as important as technical skills.

Developers often don’t to understand what's important and what's merely necessary.

- Developers tend to have a binary view of information and functionality - either it's needed or it's not.
- Instead, they should have a set of tiers or categories - critical, important, useful, rarely needed would be an example. Or even just more important | less important.

Before doing any user interface work, you should have a solid understanding of:

- What are the 4-6 most important pieces of information the user needs at this point in the work flow?
- What are the 2-3 most important actions users take at this point?

The user experience should be designed to respect the answers to those questions

Measure success with business outcomes

The idea that you have to create something to show to users instead of understanding their needs and designing to them is just wrong.

- This lesson comes from Alan Cooper

Abstract architecture isn't given enough attention.

Having "architect" in one's title doesn't mean that person knows much about architecture

My partner and I have a policy of avoiding doing work for any company that has anyone with the term "Enterprise Architect" in their title

The single greatest challenge for a developer entering the modern apps era isn't learning the technologies, or even learning UX design. It's breaking free of the past.

Change is constant, but there's regular change, and then there's "reptilian alien" change

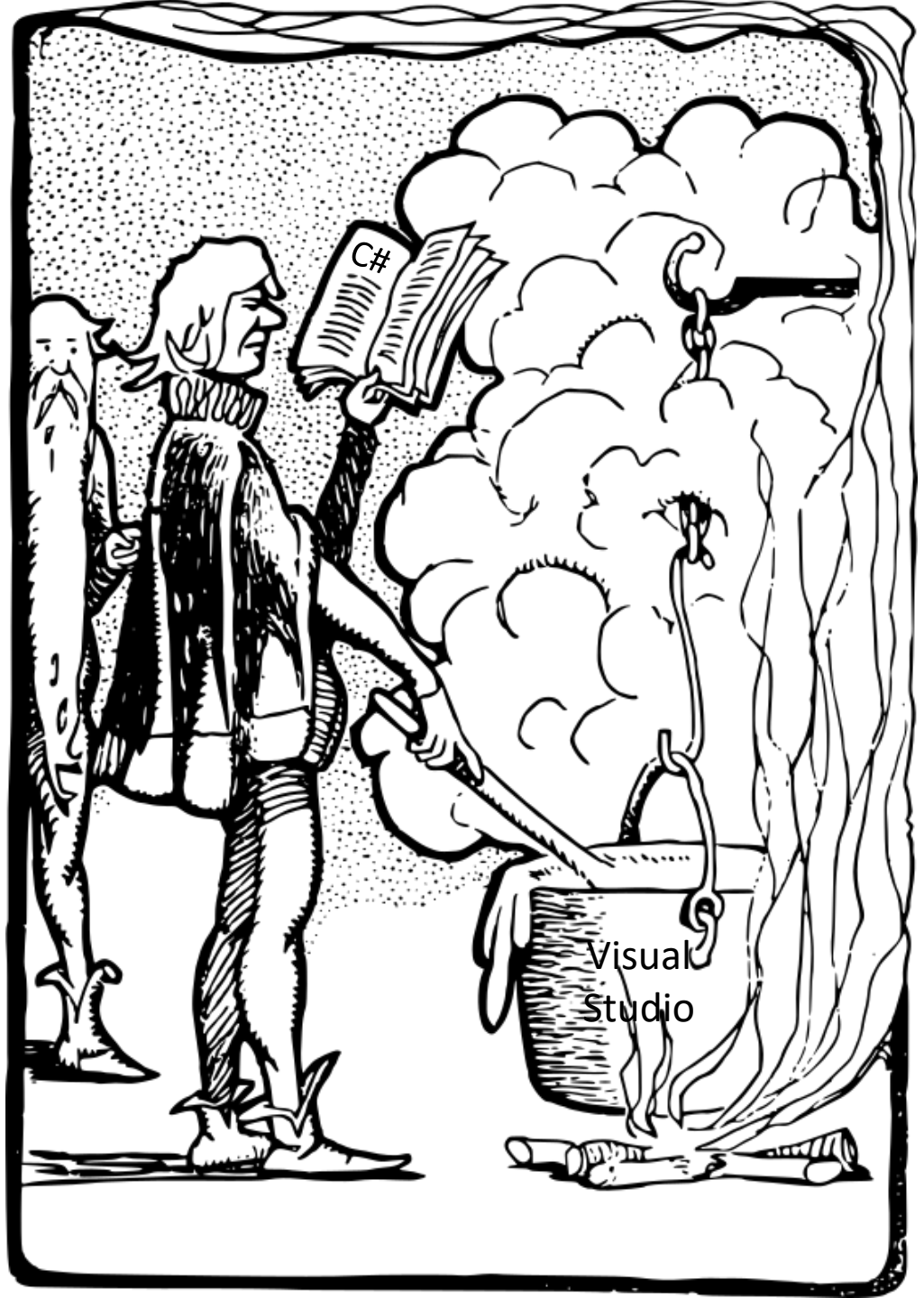
A question:

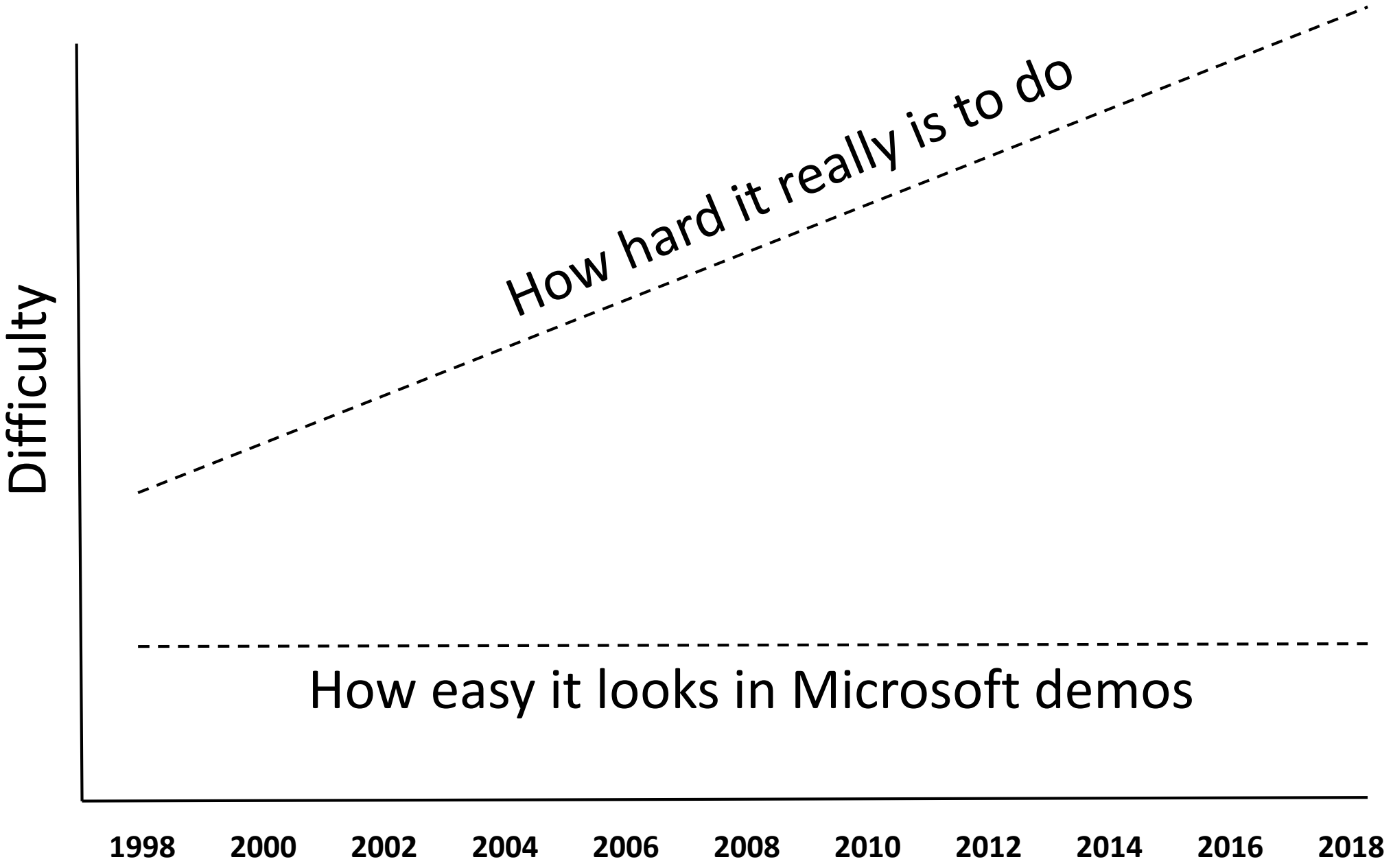
And I need an honest answer.

How many of you feel overwhelmed by the complexity of today’s application development and the constant, unrelenting change in development technologies?

If you are, I guarantee that the decision makers in your company are too

Most don’t understand what you do. You might as well be stirring a cauldron and saying magic spells.




Difficulty
How hard it really is to do
How easy it looks in Microsoft demos
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018

Angular

Dojo

Spry

Kendo

Breeze

React

Knockout

Bootstrap

JQuery

Socket.IO

Wakanda

Polymer

This has problems

- You would not be laughing at all this if it did not contain some truth
- Still skeptical? Here’s more evidence.

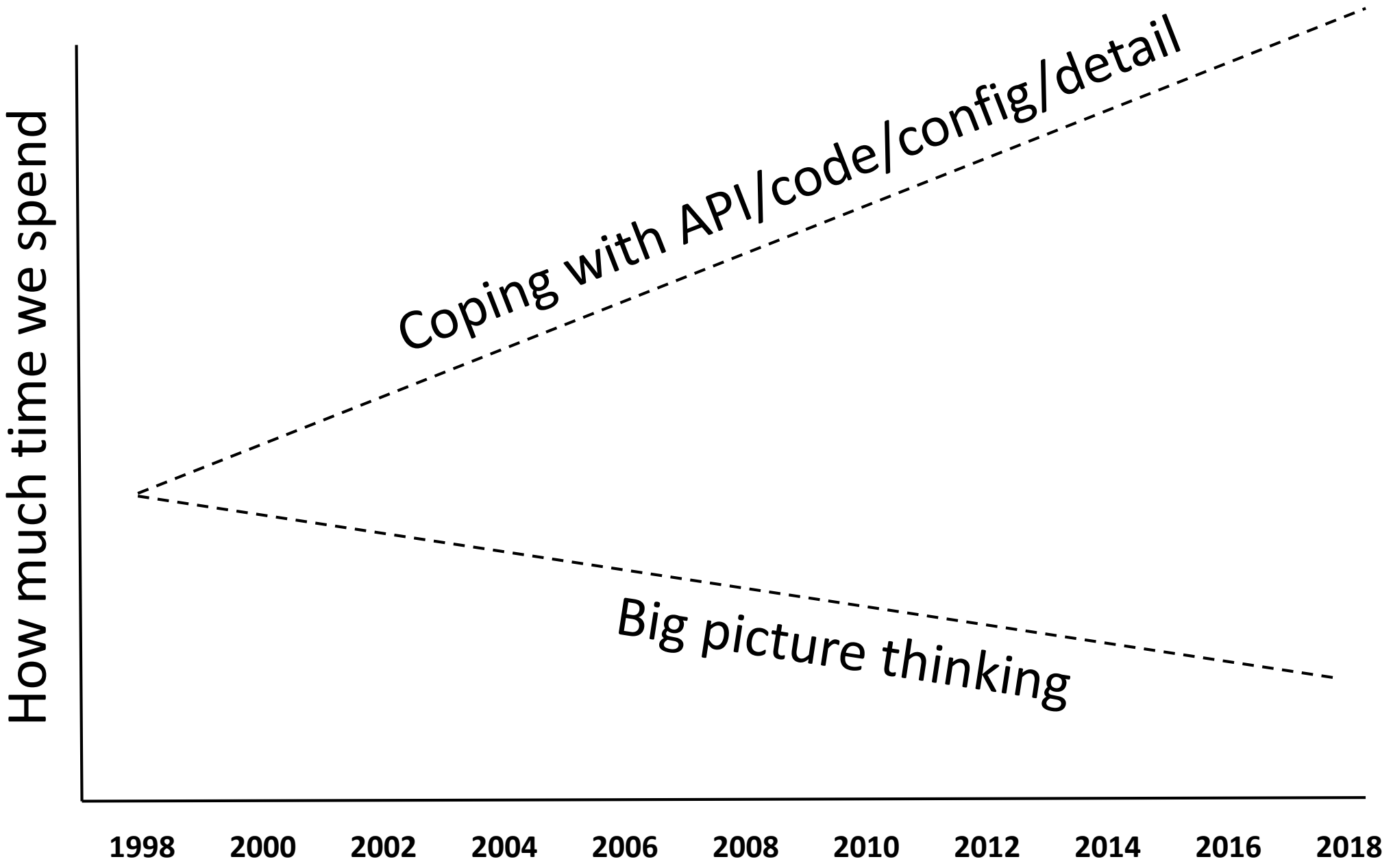
From my buddy David Neal in Atlanta

He sells these on Amazon.
He's selling a lot of them.

Or you can buy this coffee cup

We all have finite capacity.

Combined with complexity, this drives us in some negative directions


How much time we spend
Coping with API/code/config/detail
Big picture thinking
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018

A complex technology/development landscape acts as a filter – the % of big picture thinkers tends to go down over time

Among developers, work value varies a lot more than salary. Salary might go from X to 3X. But work value ranges more like X to 20X.

On the other end, "Miltons" are way too common. Some industries just won't fire them. Lack of understanding of complexity and who can cope with it is one of the big reasons why

Don't be contemptuous of management

For uncertainty, complexity, and rapid change, agile is treating the symptoms, but it can’t cure the disease

Agile is a pretty good way to make routine coding more efficient. Not the only way, but a good one.

- However, the emphasis on sprints and daily work are not a good match for certain types of innovation in software.
- Sometimes innovation requires big picture thinking, and agile's focus on the routine does not promote that.

I don't hate agile, but I don't think it's perfect, or permanent

Don't get emotionally attached to technology, process, or coding philosophy

The human mind hates uncertainty. Developers, just like other people, often make decisions with avoiding uncertainty as the main driver, and are vulnerable to those who offer solutions that sound safe.

- Sometimes the answer to “What’s the safe choice?” is “There isn’t one.”

So much of software development (as in life in general) is cost/benefit.

- Avoid using the services of anyone who does not understand that

I've listened to too many tech keynotes about cool stuff, and thought "That's fine and all, but how in the world do we make any money with this?" Too often, the answer comes two years later: we can't.

- Current offenders: AI and bots

To keep up your own value, you need a "talent stack", with talents that reinforce one another

Typical layers in the stack

- Architecture
- UX design
- Rules engines
- Workflow management
- Development Process (as long as you’re not too dogmatic or emotional about it)
- Latest technologies and newer languages
- Project management
- Useful frameworks.

Users typically outnumber developers about 50 to 1. So increasing user productivity by 2% returns as much money to the business as a 100% developer productivity increase.

Sometimes you just put up with it… but not indefinitely

Users don't trust you. They don't think you understand their work very deeply, and they're usually right about that.

Don’t put your desires and preferences ahead of the needs of the user

Stretch yourself to serve them better

Abitrary deadlines rarely work, but there's a reason managers use them.

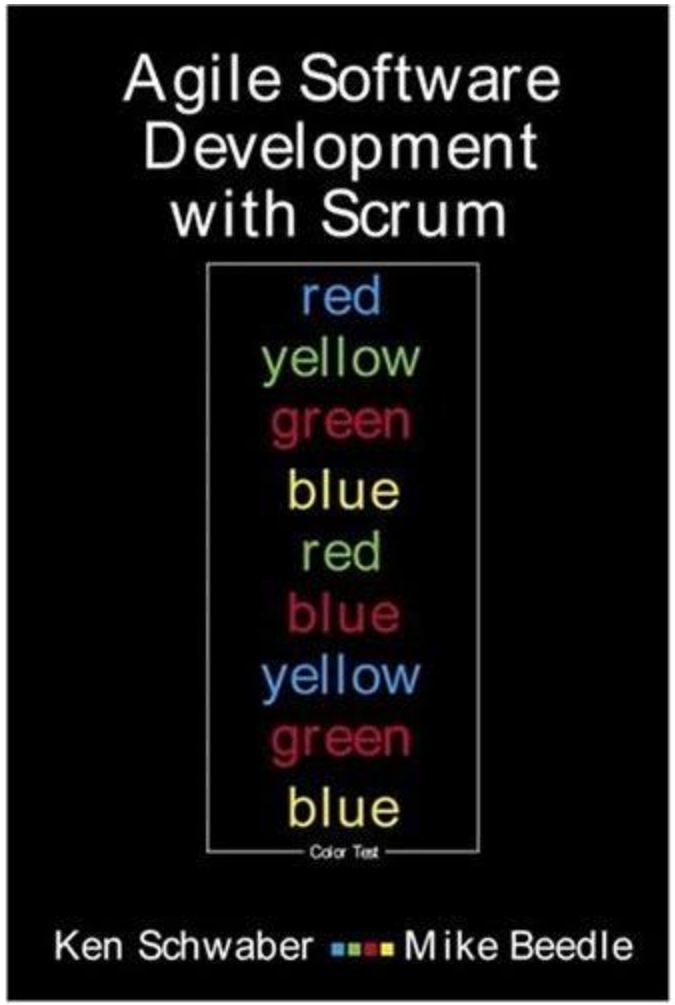
In the Microsoft space, it helps to understand the strengths and weaknesses of Microsoft people.

Don't take responsibility for other people's messes

Saying "this works for me, you should try it" is fine. Saying "this works for me, it will work for anyone, and everyone should use it, or they're backward and ignorant" is not fine

- Avoid taking advice from anyone who talks like that
- For a great example, see the first page of the seminal book on Scrum

“Industrial process control theory is a proven body of knowledge that describes why Scrum works and other approaches are difficult and finally untenable.”



Beware of clever sounding glibness

Superficial analysis

“There’s no such thing as the cloud.
It’s just someone else’s computer.”

Equivalent logic

“There’s no such thing as a restaurant. It’s just someone else’s kitchen.”

Someday, odds are about 50 to 1 that you'll hit the wall and need a radical change in your career

- User interface design and prototyping
- Native app development – UWP/WPF/XAML
- Training on user experience design
- Training on XAML
  - Windows 10 / WPF / mobile / touch
  - Beginning through advanced
- Mentoring on XAML and UX design

Billy Hollis

www.nextver.com for samples and videos

billyhollis@live.com or billyhollis@gmail.com