ACRUMEN:

What IS Software Quality, Anyway?!

by Dave Aronson
T.Rex-2025@Codosaur.us

CODOSAURUS

www.Codosaur.us

@DaveAronson

NOTE TO SELF: have biz card ready to show, and more as handouts!

Current time: ~77 mins speaking at mixed rate with frequent water-breaks; slot is 90, w/ 5-10 for Q&A, so want 80-85! REMEMBER TO SLOW DOWN; maybe add more stuff.

ACRUMEN:

What IS Software Quality, Anyway?!

by Dave Aronson

T.Rex-2025@Codosaur.us

CODOSAURUS

www.Codosaur.us

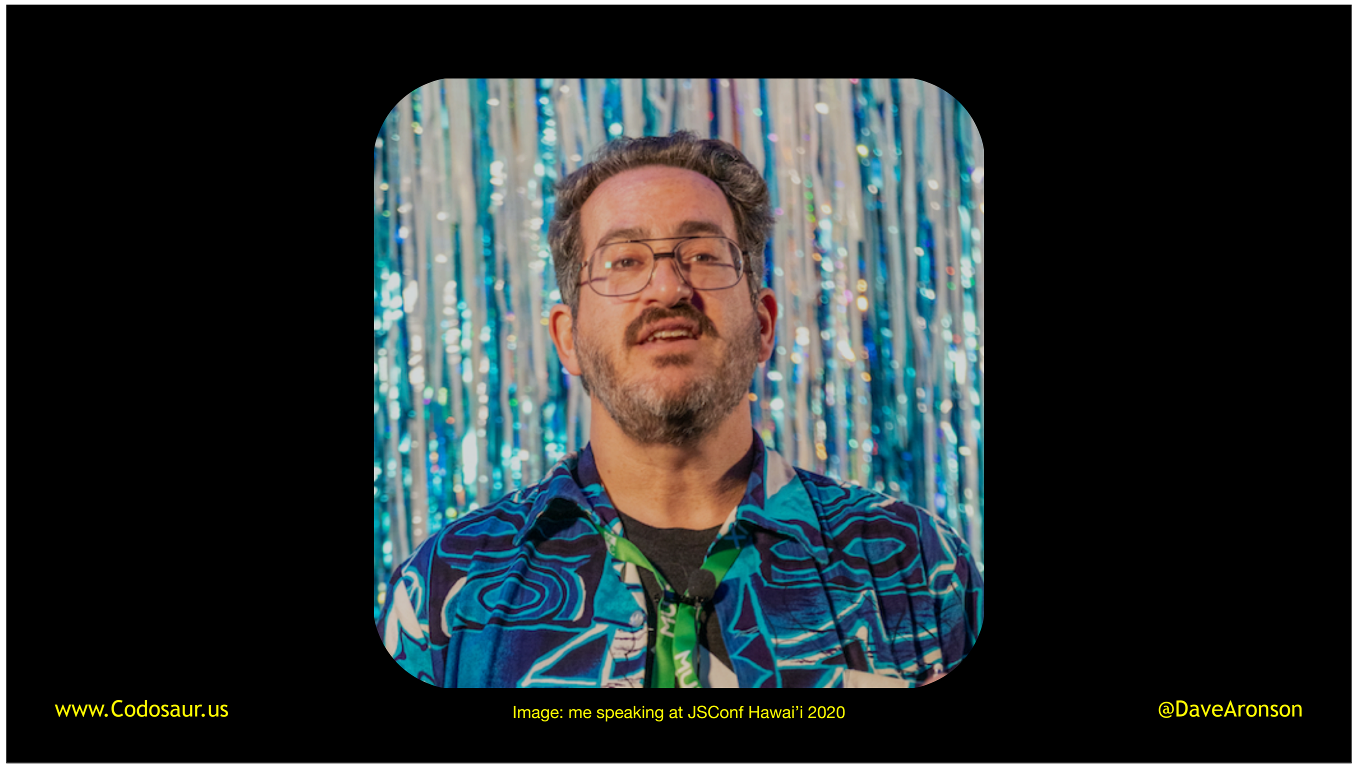
@DaveAronson

(Duplicate slide so I can flip to a new one to start my timer, ignore this.)

Hello,

www.Codosaur.us Image: https://www.pexels.com/photo/london-from-the-river-27585634/ @DaveAronson

London! I'm



www.Codosaur.us

Image: me speaking at JSConf Hawai'i 2020

@DaveAronson

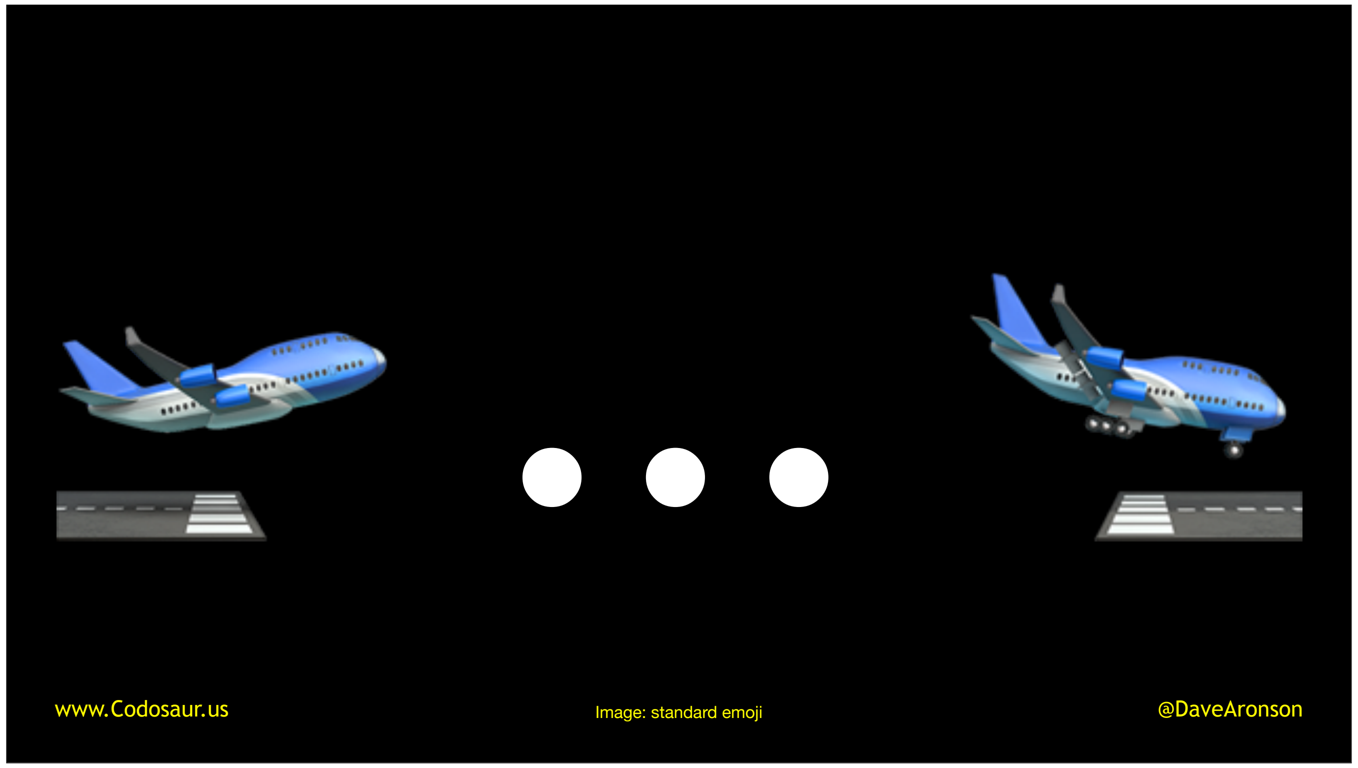
Dave Aronson, the

www.Codosaur.us

Image: my company logo!

@DaveAronson

Tyrannosaurus Rex of Codosaurus, and I



flew all the way over here, on my

www.Codosaur.us

Images: https://pixabay.com/vectors/dinosaur-tyrannosaurus-t-rex-6273164/
and https://pixabay.com/vectors/bird-flying-wings-dinosaur-ancient-44859/

@DaveAronson

pet pterodactyl, to

tell you all about

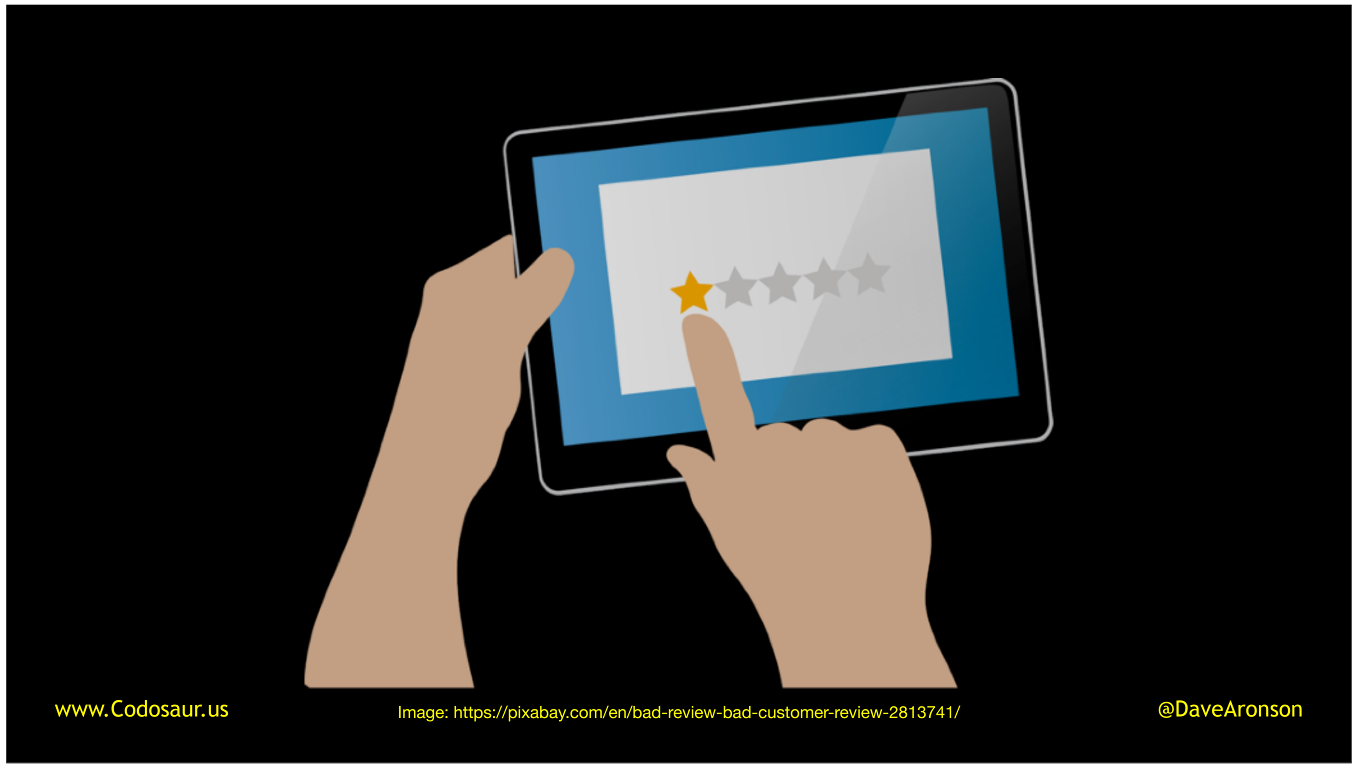
ACRUMEN

www.Codosaur.us

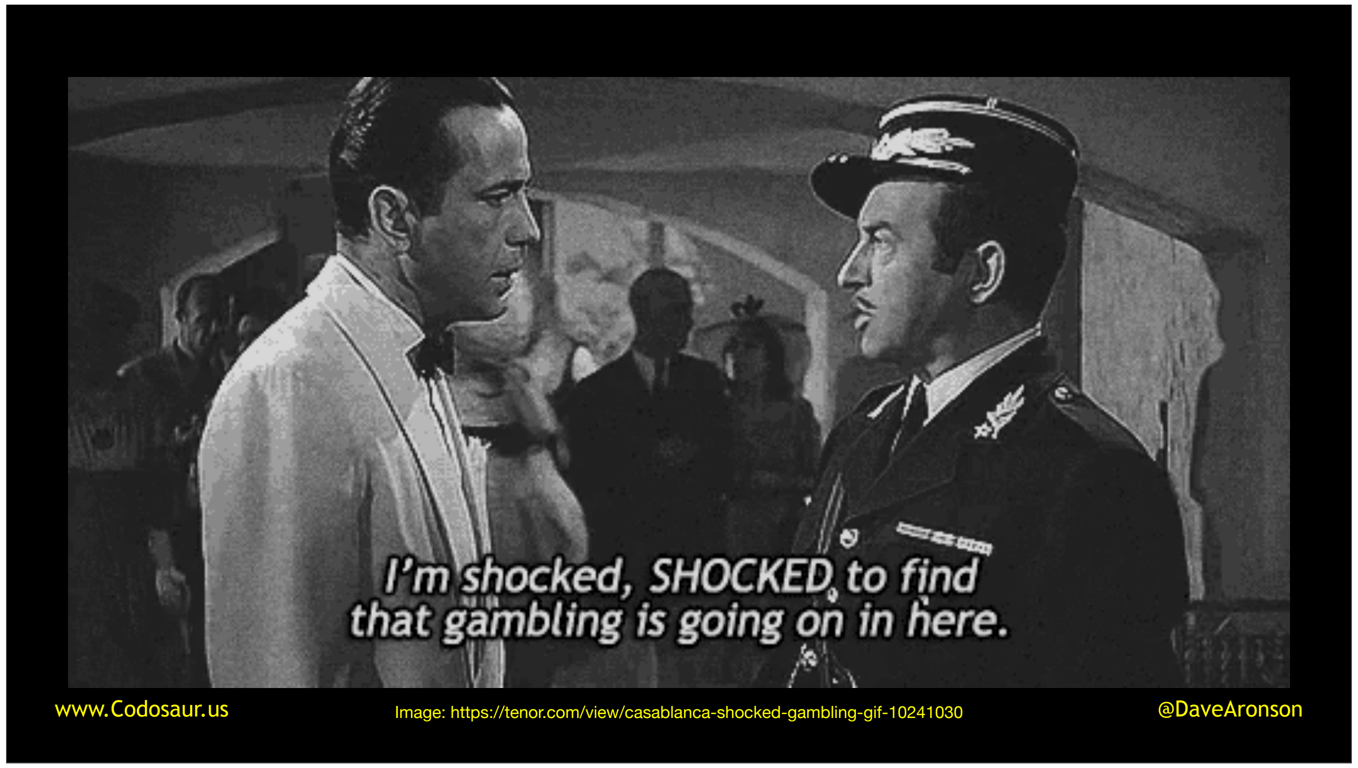
@DaveAronson

ACRUMEN, my free, low-jargon, technology-agnostic, brief, simple, yet more or less comprehensive, definition of software quality. And, some tips to help you achieve it.

Let’s start things off with a question. Do you like . . .


www.Codosaur.us
Image: https://pixabay.com/en/bad-review-bad-customer-review-2813741/
@DaveAronson

. . . low quality software? (pause) Anybody? Anybody? Nobody?! I'm . . .


I'm shocked, SHOCKED to find
that gambling is going on in here.
www.Codosaur.us
Image: https://tenor.com/view/casablanca-shocked-gambling-gif-10241030
@DaveAronson

shocked, SHOCKED!

Okay, let's try another question. Have you . . .

www.Codosaur.us
Image: https://www.publicdomainpictures.net/en/view-image.php?image=174322
@DaveAronson

. . . *written* any low quality software? (raise hands) Yeah, I know I sure have!

That's more like it! To those of you who said yes, congratulations! As the saying goes, Step One is to realize . . . you have a problem! For the rest of you: welcome to software development! I hope you enjoy this career you've obviously just started.

So we've got a lot of people writing (or at least having written) low-quality software, but we don't *like* it. It seems pretty clear to me, we need . . .

. . . more cowbell, er, I mean, . . .

www.Codosaur.us

Image: https://pixabay.com/en/software-packaging-quality-500956/

@DaveAronson

. . . more software quality!

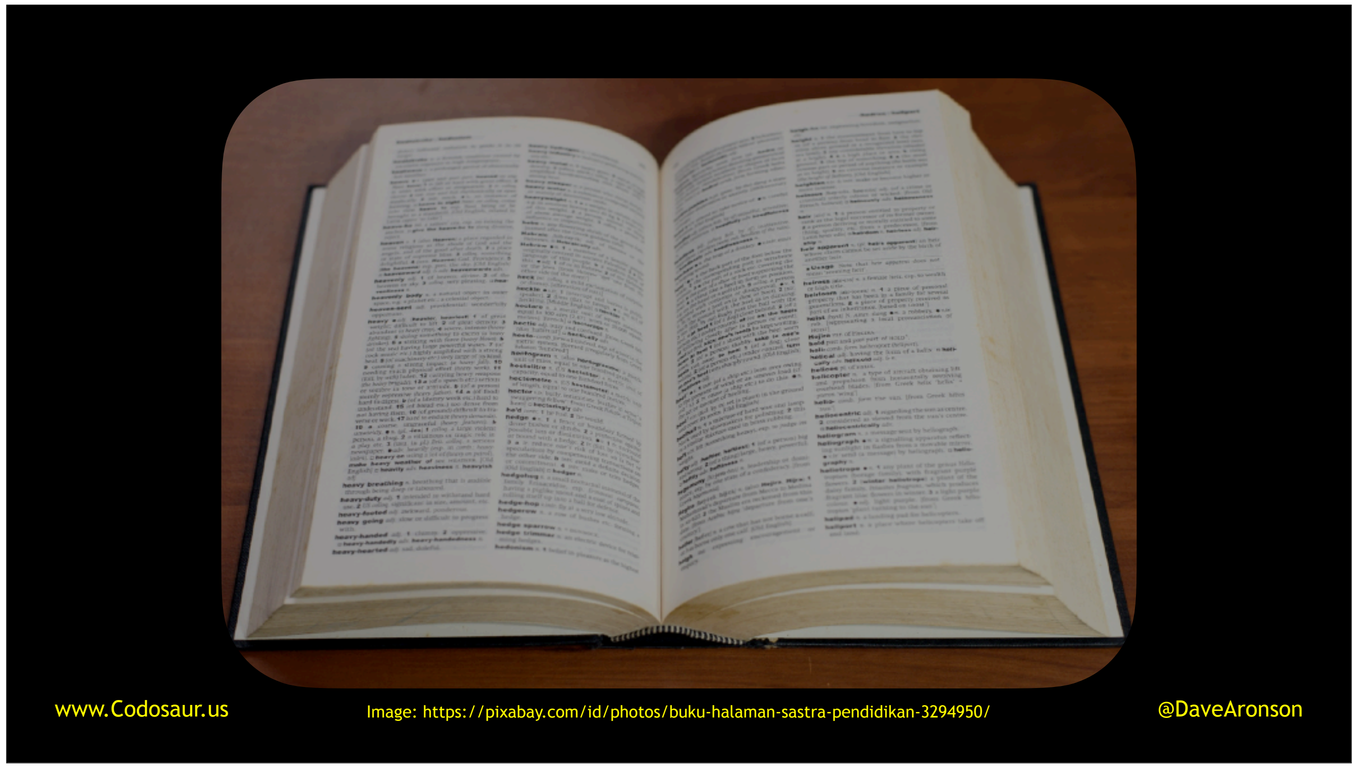
But that just leads us to one tiny little question: . . .

www.Codosaur.us

Image: https://www.pexels.com/photo/brown-monkey-photo-767197/

@DaveAronson

. . . What *is* it? Without a good . . .

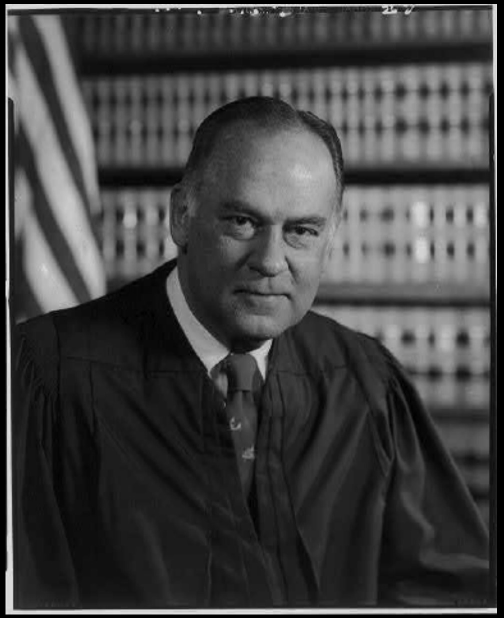

www.Codosaur.us
Image: https://pixabay.com/id/photos/buku-halaman-sastra-pendidikan-3294950/
@DaveAronson

. . . *definition*, it's very difficult to achieve — we can't hit a nonexistent target! That makes it very hard to improve even our *own* software quality, let alone the entire state of the art! And if we don't *share* that definition with other people, it's very hard to get them to *acknowledge* when we *have* achieved it — or at least we *think* we have. So, I'm trying to get everybody on the same page with this. (Yeah, I know, good luck with that!)

But even if you don't . . .

www.Codosaur.us
Image: https://www.flickr.com/photos/rikkis_refuge/4282810209
@DaveAronson

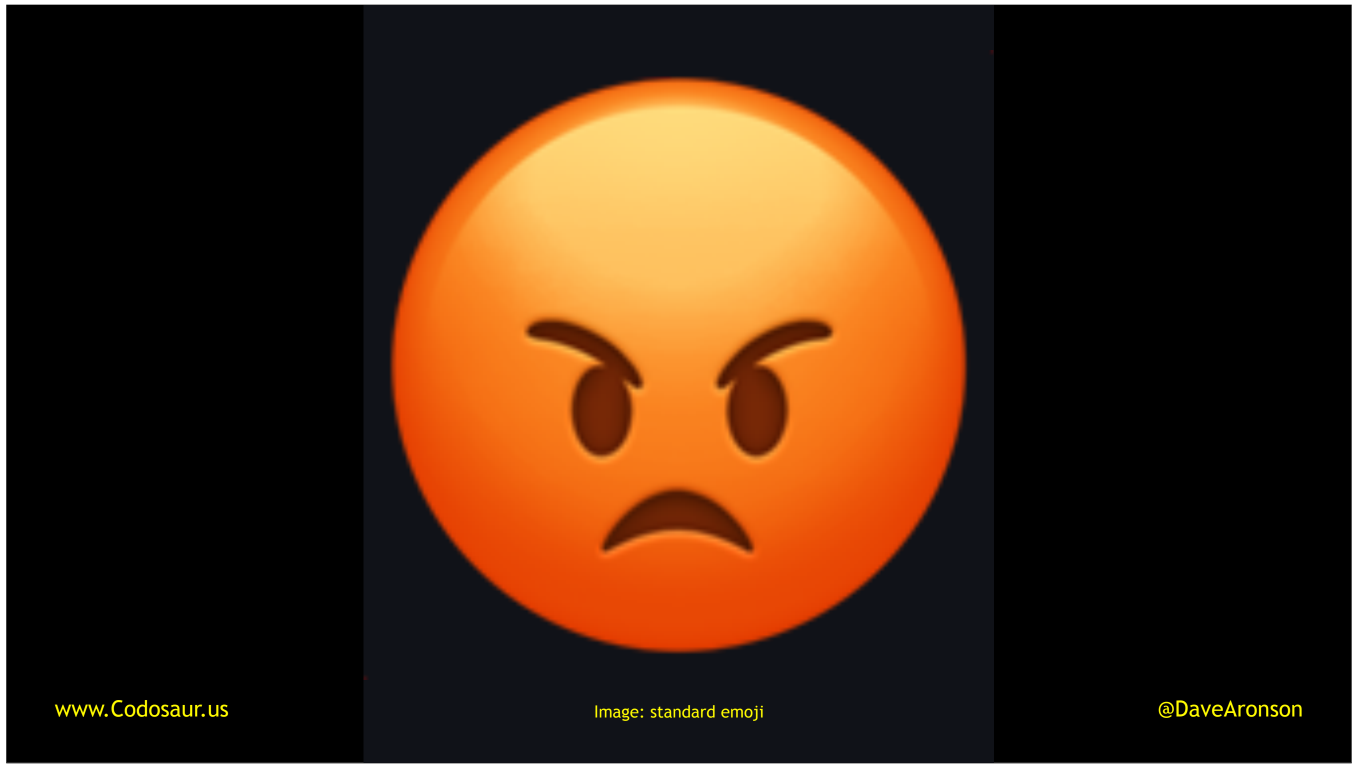
. . . adopt *my* definition, if you at least adopt *some* reasonable definition, that's *still* better than the current situation, where we tend to toss this word "quality" around with *little to no* definition! We may take an approach like American Supreme Court Justice . . .



www.Codosaur.us Image: https://picryl.com/amp/media/official-portraits-of-the-1976-us-supreme-court-justice-potter-stewart @DaveAronson

. . . Potter Stewart. In a 1964 court case, he could not come up with a solid definition of hard-core pornography, so do any of you know what he famously said? (WAIT FOR ANSWER) He said, and you've probably heard this quoted, "I know it when I see it." But I would argue that we really *don't* know software quality when we see it! In fact, quite the reverse, we *mainly* know software quality when we *don't* see it. When was the last time you heard anybody say . . .

. . . "this program does just what I need, runs so fast, and is so easy to use, and always gives me perfect results"? Probably no more than once or twice a year, including when they're talking about someone *else's* software, not just bragging about their own! But what about hearing people say . . .



. . . “this damn slow buggy piece of junk is so hard to use and doesn’t even do what I really need! Grrrrr!”? For some of us, that’s daily life — not only hearing other people say it, but saying it ourselves! And for the more technical aspects, even working on a codebase we find easy to change, is generally not cause for any comment, while working on a difficult one, draws endless complaint. Humans just *love* . . .

Image: https://getyarn.io/yarn-clip/2fa416bc-4803-480a-bcc4-0826dc51ef10/gif

. . . to complain.

Before we delve into the definition, though, I'd like to level-set some . . .

www.Codosaur.us

Image: https://picryl.com/media/great-expectations-1917-film-683b78

@DaveAronson

. . . expectations. This talk is aimed mainly at intermediate and perhaps some advanced junior developers. Intermediates have generally realized how important yet . . .

www.Codosaur.us
Image: https://stockcake.com/i/cuddly-teddy-bear_1522870_1173649
@DaveAronson

. . . fuzzy software quality is. Juniors usually haven't, but if you're a junior and you're there, or at least interested, that's great! (Maybe you're not really a junior any more!) Seniors can also make good use of this. They have usually already developed (no pun intended) their own approach to quality. However, they may be struggling to put it into words, so as to teach quality to any intermediates and juniors they oversee or mentor. This approach might not match theirs, but at least it could be *some* useful approach to teach them.

I'd also like to set the . . .

. . . question policy. If you have a question that absolutely must be answered right now, in order for you to understand some important point I'm making, raise a hand, and I'll try to notice and call on you. If you have a less urgent question, please save it for the end, and save any comments for after all the questions.

And lastly, this isn't meant for the . . .

www.Codosaur.us

Image: https://pxhere.com/en/photo/170658

@DaveAronson

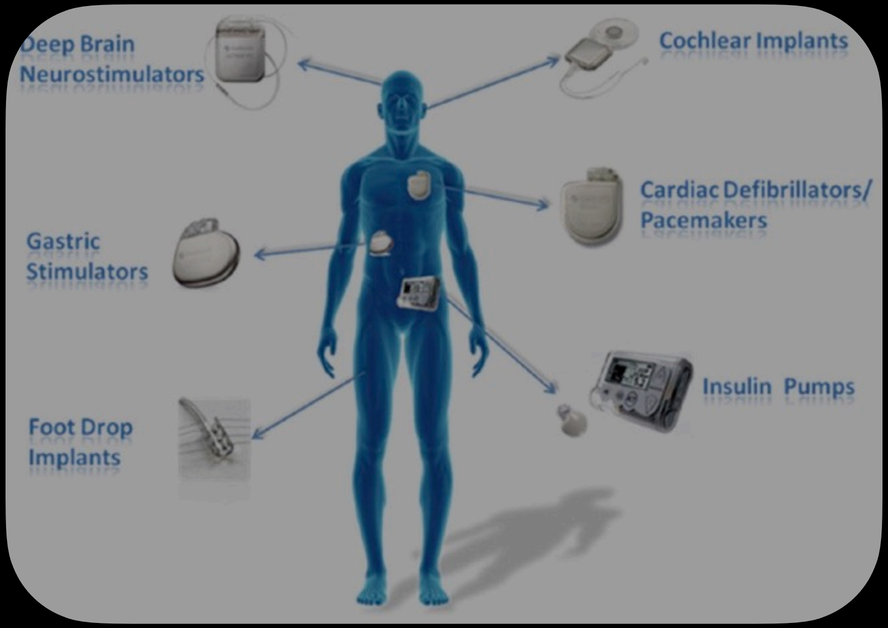
. . . *extreme* levels of quality, mainly in the robustness, usually associated with the software used in . . .

www.Codosaur.us

Image: https://www.flickr.com/photos/vintage_illustration/44546915360

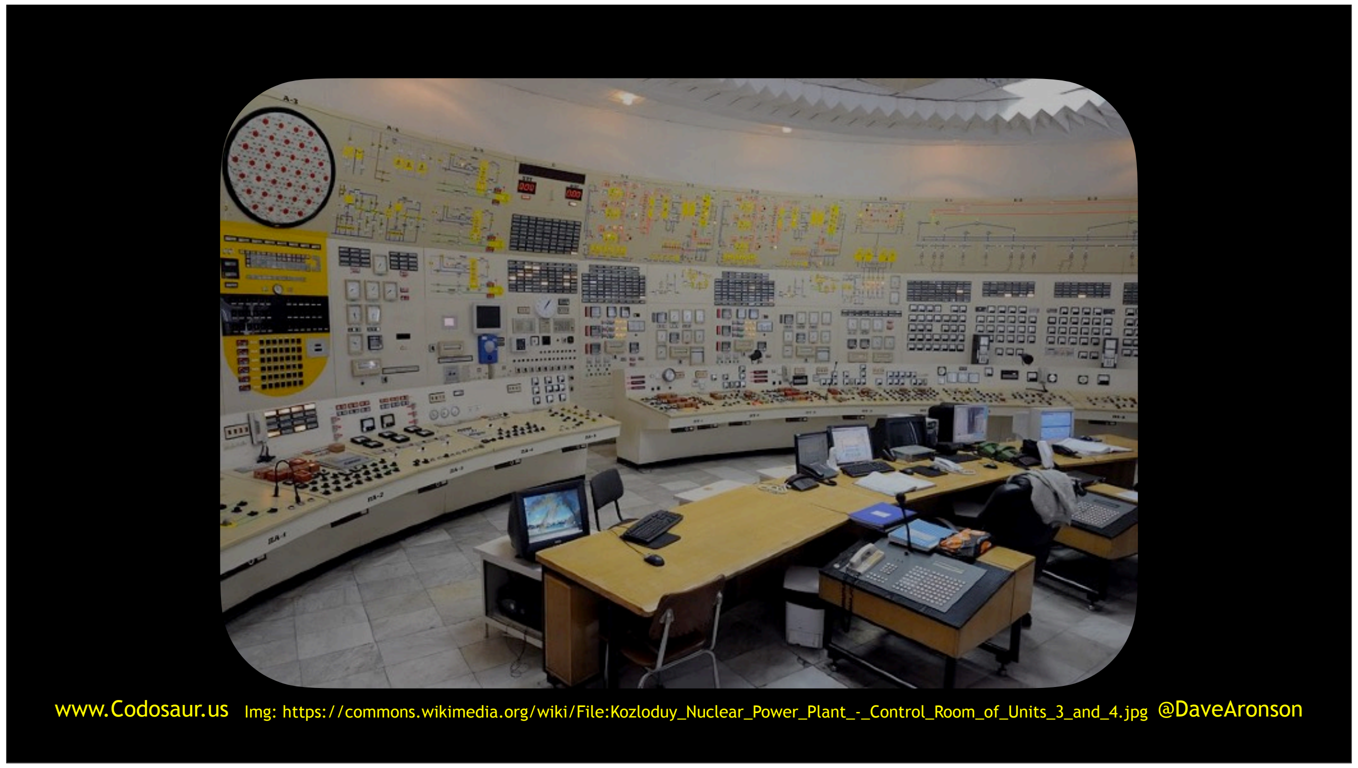
@DaveAronson

. . . avionics, . . .



www.Codosaur.us Image: https://www.researchgate.net/figure/Common-implantable-devices_fig10_332730830 @DaveAronson

. . . implanted medical devices, . . .


www.Codosaur.us
Img: https://commons.wikimedia.org/wiki/File:Kozloduy_Nuclear_Power_Plant_-_Control_Room_of_Units_3_and_4.jpg
@DaveAronson

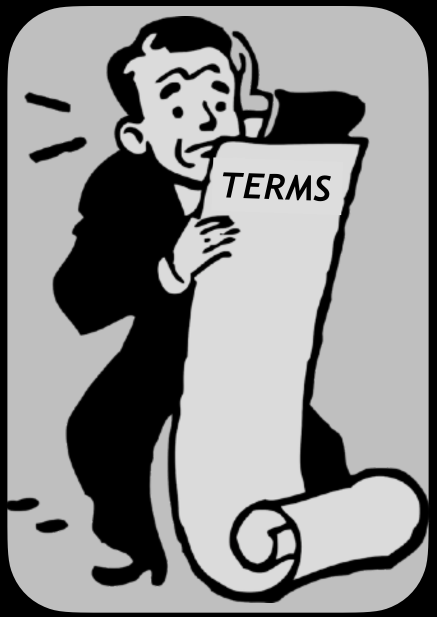
. . . nuclear powerplants, or even . . .

www.Codosaur.us Image: https://picryl.com/media/first-bank-of-the-united-states-philadelphia-pennsylvania-1 @DaveAronson

. . . banks, where correctness is of paramount importance. It's meant for the other five-nines of us, writing consumer-grade systems like typical web or mobile apps, or maybe internal business apps. So, if something goes wrong, there may be frustration on the users' part and embarrassment on ours, but nobody's going to *die*, or even lose a lot of money. Those other kinds of industries already have their own approaches, often including regulations, and *much* . . .

. . . closer *inspection* than *our* software will ever get.

So, back to actually talking about the definition. With all these examples around, of defining *very* high quality, why do we need a *new* definition? Several years ago, I was *looking* for a good *general-purpose* definition, but everything I found had some serious problems, at least for this purpose. Most were . . .



www.Codosaur.us Image: https://pixabay.com/id/vectors/khawatir-membayar-uang-kertas-pria-30148/ + my text @DaveAronson

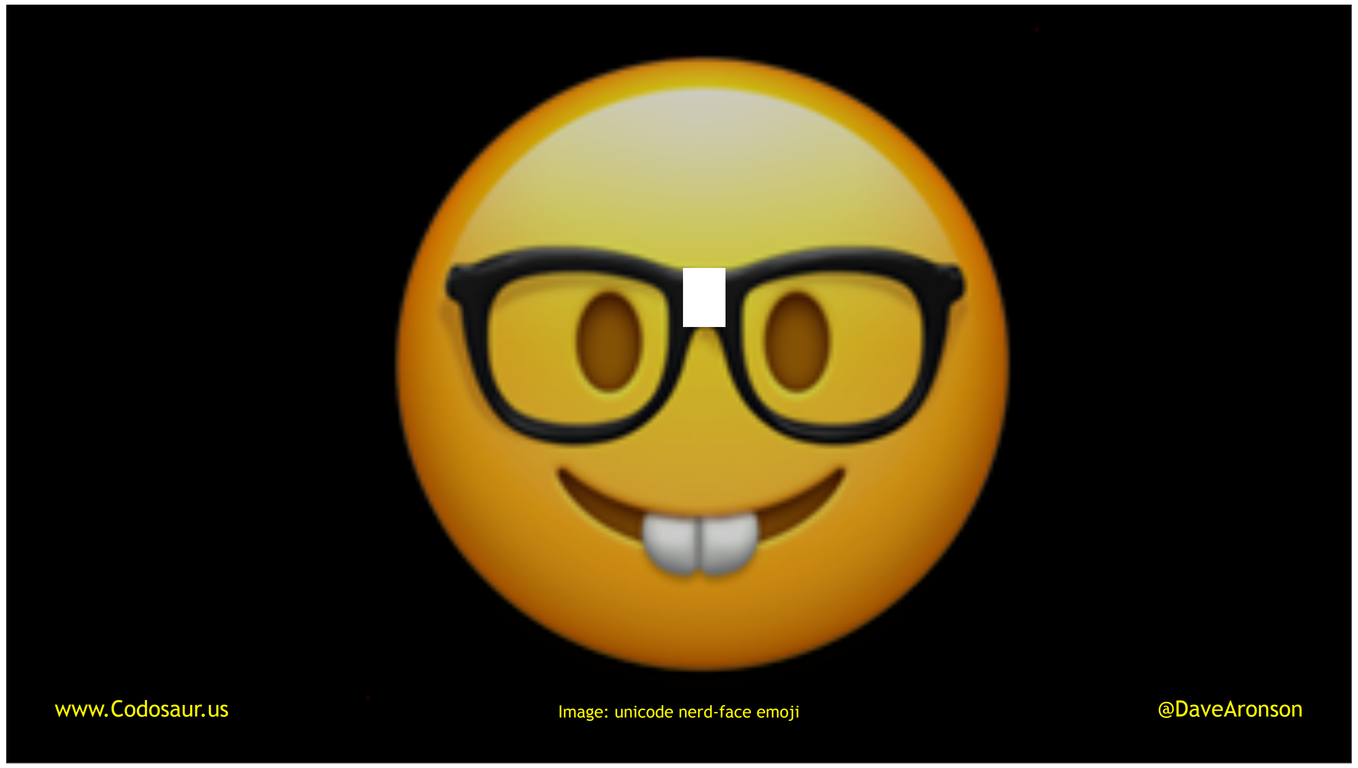
. . . long lists of complicated terms, full of developer jargon. Now don't get me wrong, jargon is fine for talking amongst *ourselves,* but I wanted a definition that *other* people would understand, *even non-technical people*, so they could have a better understanding of the *challenges* we developers face, and they could give us more precise feedback about *exactly how* our software sucks. (And you know most of it does, right?) Also, I wanted something *short* enough, and *simple* enough, that people could easily learn it, remember it, and apply it, without having to consult lengthy and complex documentation.

Some definitions were . . .

www.Codosaur.us
Image: https://www.flickr.com/photos/59937401@N07/5857412037
@DaveAronson

. . . proprietary, requiring us to buy expensive software tools, or at least relatively expensive documentation — like I just said I didn't want people to have to use. Some were only applicable within the context of certain technologies, often also proprietary, or certain styles of programming, like Object-Oriented or Functional or whatever. Some were even limited in the type of software, like being only about *web* applications, or *mobile* apps. But I wanted something we could use with *all* software, for *free*.

Some definitions focused exclusively on issues of interest to . . .



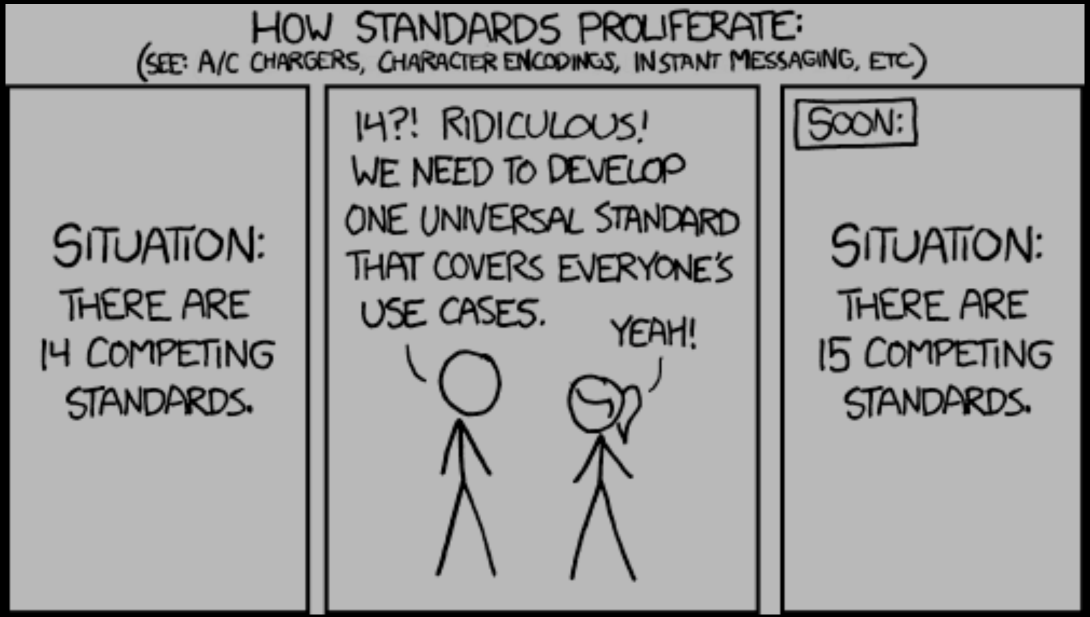
. . . us developers, ignoring the needs of the users and other stakeholders. For instance, many were completely about maintainability, which *nobody else knows or cares about*, at least not *directly*, and omitted other important things, like whether the software is *easy to use*, or even gives the right answers! Management may know and care about the *effects* of poor maintainability, such as changes taking longer and introducing bugs, but they generally *don't know* that these *symptoms* are *caused* by poor *maintainability*.

Some definitions weren't even really about the software at all, but about the process or its byproducts, dictating that we must . . .

www.Codosaur.us
Images: https://pixabay.com/illustrations/meeting-conference-personal-3321175/
and many copies of https://pixabay.com/vectors/papers-stack-heap-documents-576385/
@DaveAronson

. . . hold these meetings or write those documents, even aside from the documentation-heavy processes used in those safety-critical industries. Now, some of these meetings and documents may be *helpful*, but to make them the actual *definition*, I felt completely missed the point. It's certainly possible to do all that, and produce horrible software anyway, or to produce great software *without* any of that overhead. I wanted something more flexible, *de*scriptive rather than *pre*scriptive, and more focused on the software itself.

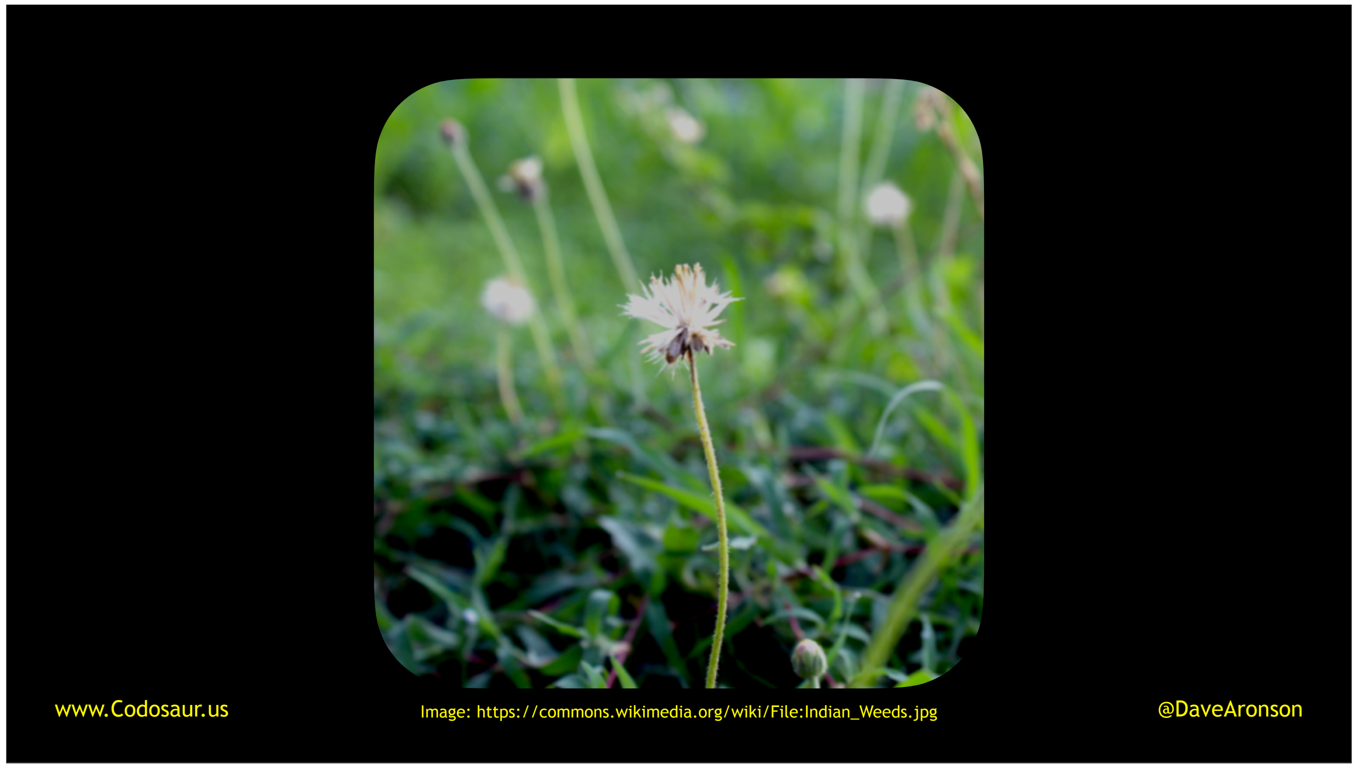
Long story short, okay, maybe it’s too late for that, so long story *medium*, I didn't see anything that I liked, nor that was commonly accepted, so in the spirit of . . .


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

www.Codosaur.us Image: from https://xkcd.com/927/ (license: https://creativecommons.org/licenses/by-nc/2.5/) @DaveAronson

. . . XKCD (PAUSE!), I decided to make my own. But rather than the canonical approach of taking the best parts of all of the existing definitions, I decided to whittle it down to just the bare essentials.

To keep it simple, I (step back) zoomed out from . . .



www.Codosaur.us

Image: https://commons.wikimedia.org/wiki/File:Indian_Weeds.jpg

@DaveAronson

. . . down in the weeds, where we developers tend to live, past the . . .

www.Codosaur.us Image: https://www.flickr.com/photos/158652122@N02/44921371014 @DaveAronson

. . . 10,000 meter view, up to about . . .

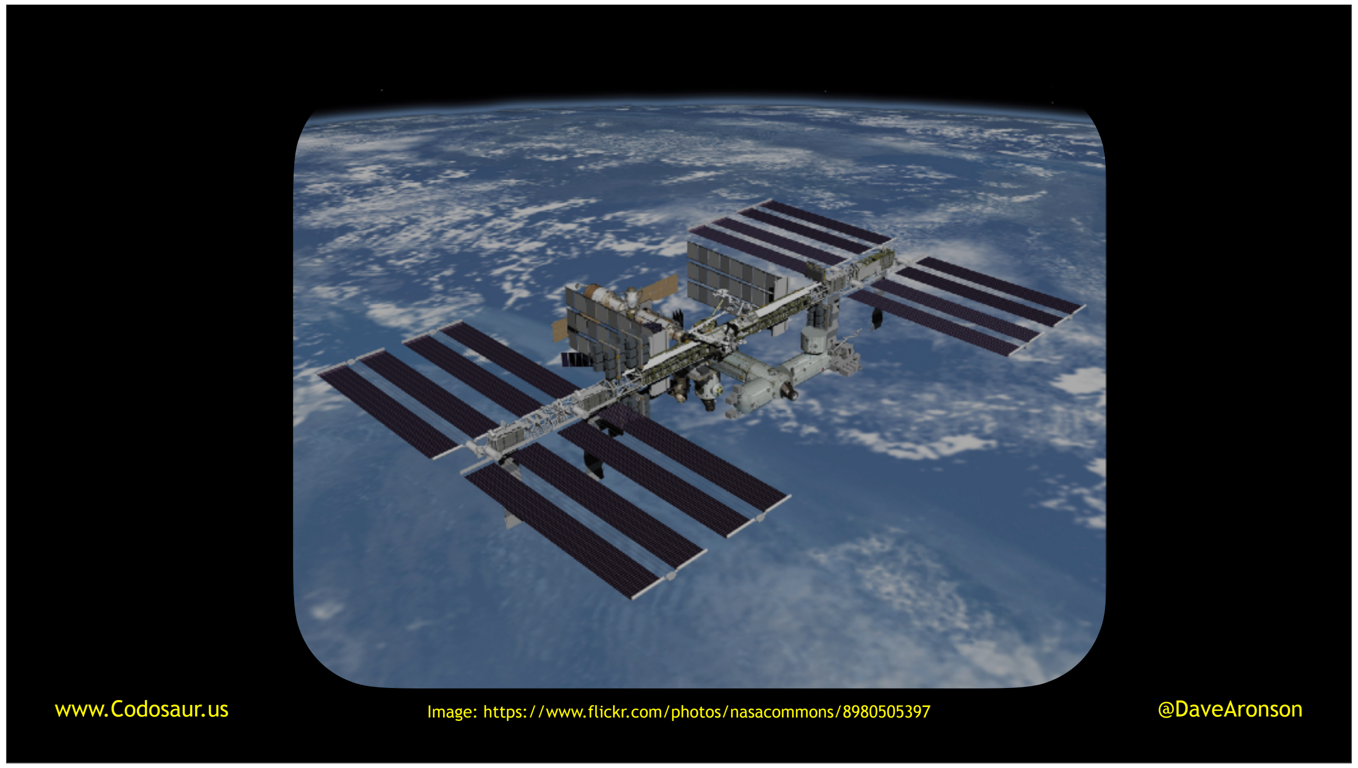
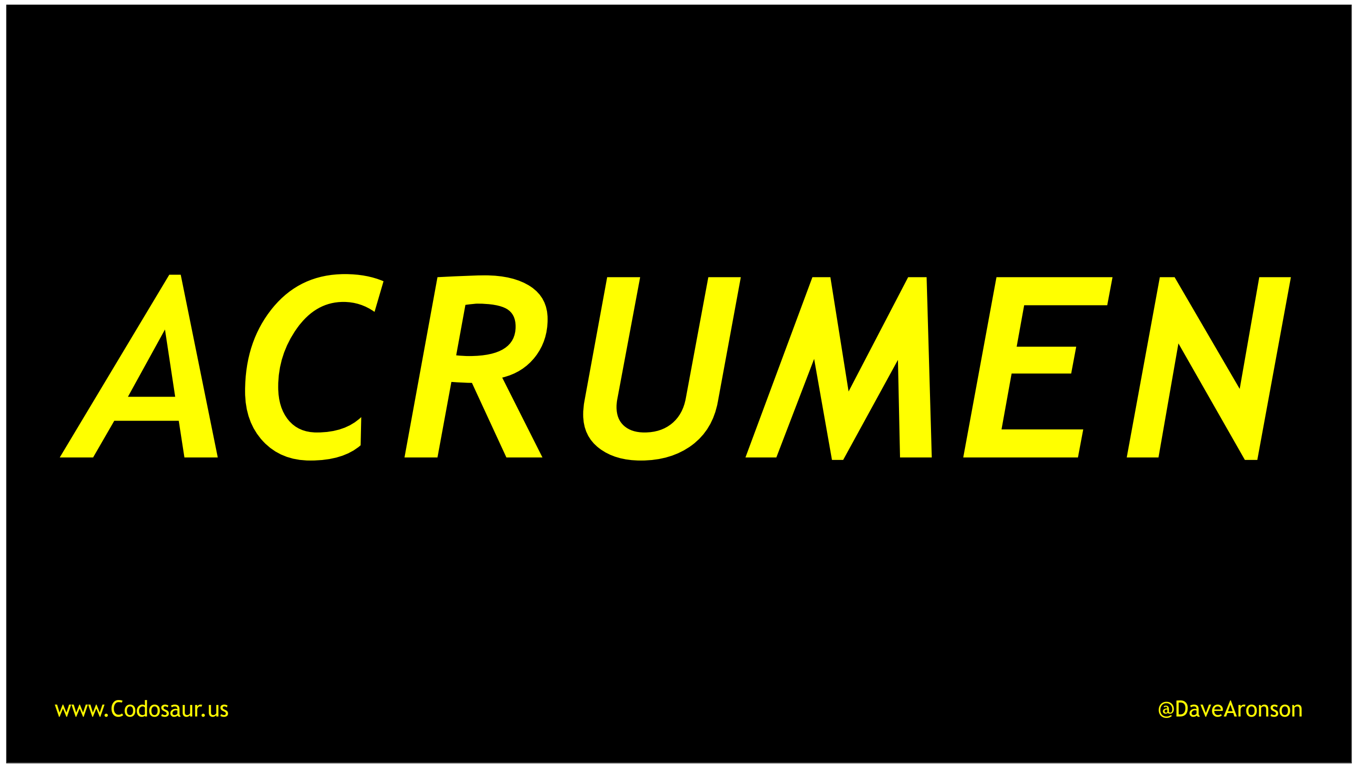

www.Codosaur.us

Image: https://www.flickr.com/photos/nasacommons/8980505397

@DaveAronson

. . . low earth orbit, so I could look at continents, not pebbles. That let me trim it down to a list of just six aspects, with simple names and *relatively* simple explanations. The result is so short that, even with a basic explanation, it literally fits on the back of a business card, without even using small print, and (HOLD UP BIZ CARD!) here's mine to prove it. See me later if you want one as a cheat-sheet.

I call this list of aspects . . .



ACRUMEN

www.Codosaur.us

@DaveAronson

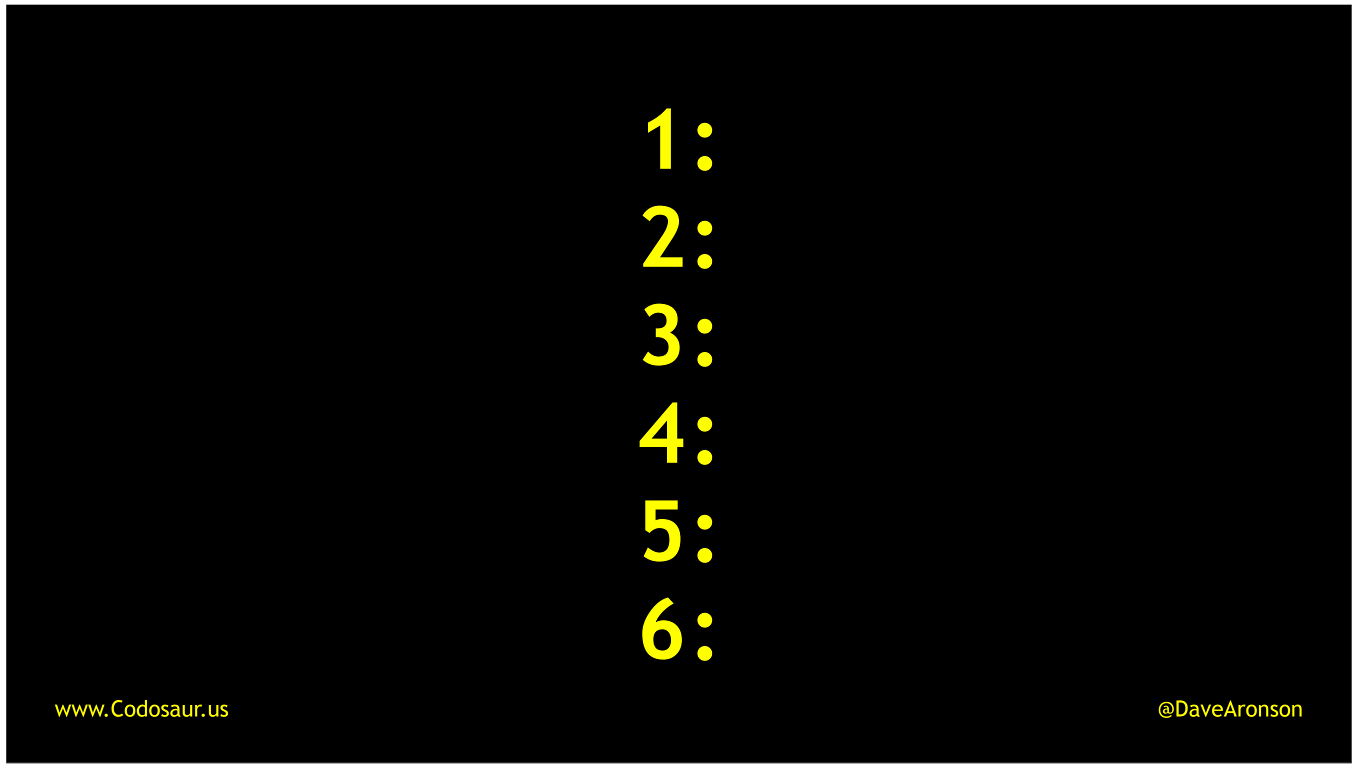
. . . ACRUMEN, but what does that mean? Originally, that was a Latin word, meaning sour fruit, like grapefruits, limes, and of course especially . . .

www.Codosaur.us
Image: https://pixabay.com/en/lemonade-lemons-glass-beverage-1447521/
@DaveAronson

. . . lemons. That's why you'll see lots of bright lemon yellow throughout these slides. It's basically "The Official Color of ACRUMEN".

So, I'm taking those sour lemons that life hands us, often by the bushelful, sometimes in the form of low-quality software, and making what I hope will be delicious lemonade, in the form of the ACRUMEN software quality definition. Which finally brings us to the question, what is *that*?

The *acronym* ACRUMEN (try saying that ten times fast!), simply takes those six aspects, and . . .



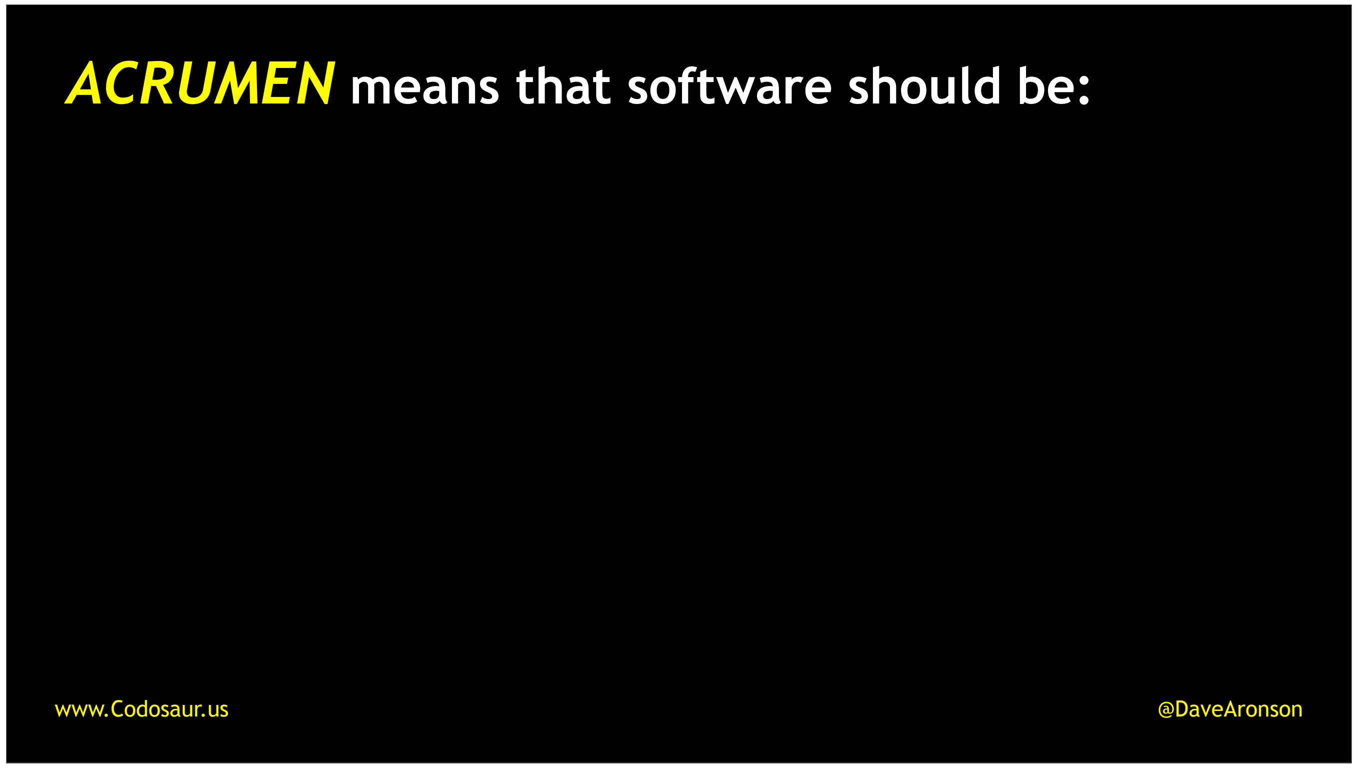
1:
2:
3:
4:
5:
6:

www.Codosaur.us

@DaveAronson

. . . puts them in priority order.

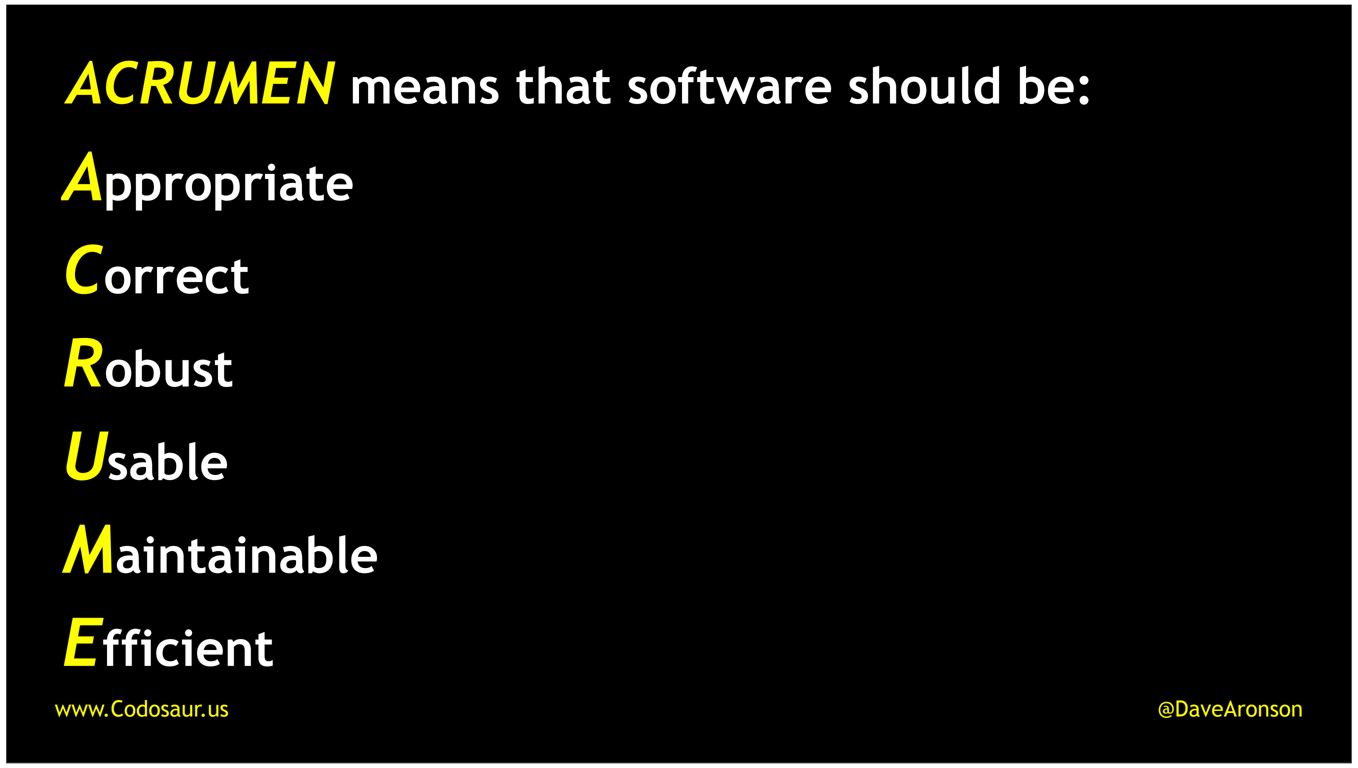
By now you're probably wondering, SO WHAT ARE THE BLANKETY-BLANK ASPECTS ALREADY?! They are that . . .



ACRUMEN means that software should be:

www.Codosaur.us @DaveAronson

. . . software should be: (INHALE) . . .



ACRUMEN means that software should be:

*A*ppropriate

*C*orrect

*R*obust

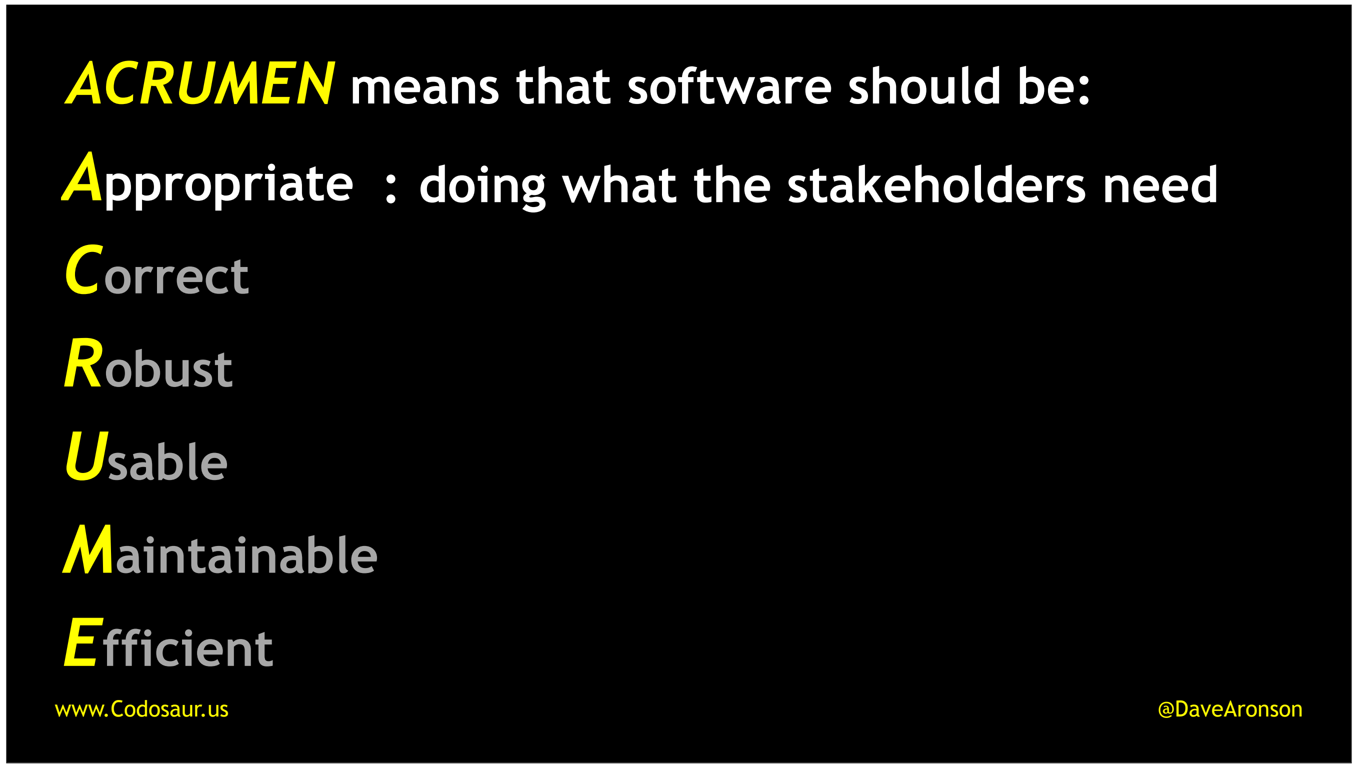
*U*sable

*M*aintainable

*E*fficient

www.Codosaur.us @DaveAronson

. . . Appropriate, Correct, Robust, Usable, Maintainable, and Efficient, usually in that order. But what does all *that* mean? From the amount of space left on this slide, you've probably already guessed that I'm going to fill it in with explanations. First and foremost, software needs to be . . .



ACRUMEN means that software should be:

*A*ppropriate : doing what the stakeholders need

*C*orrect

*R*obust

*U*sable

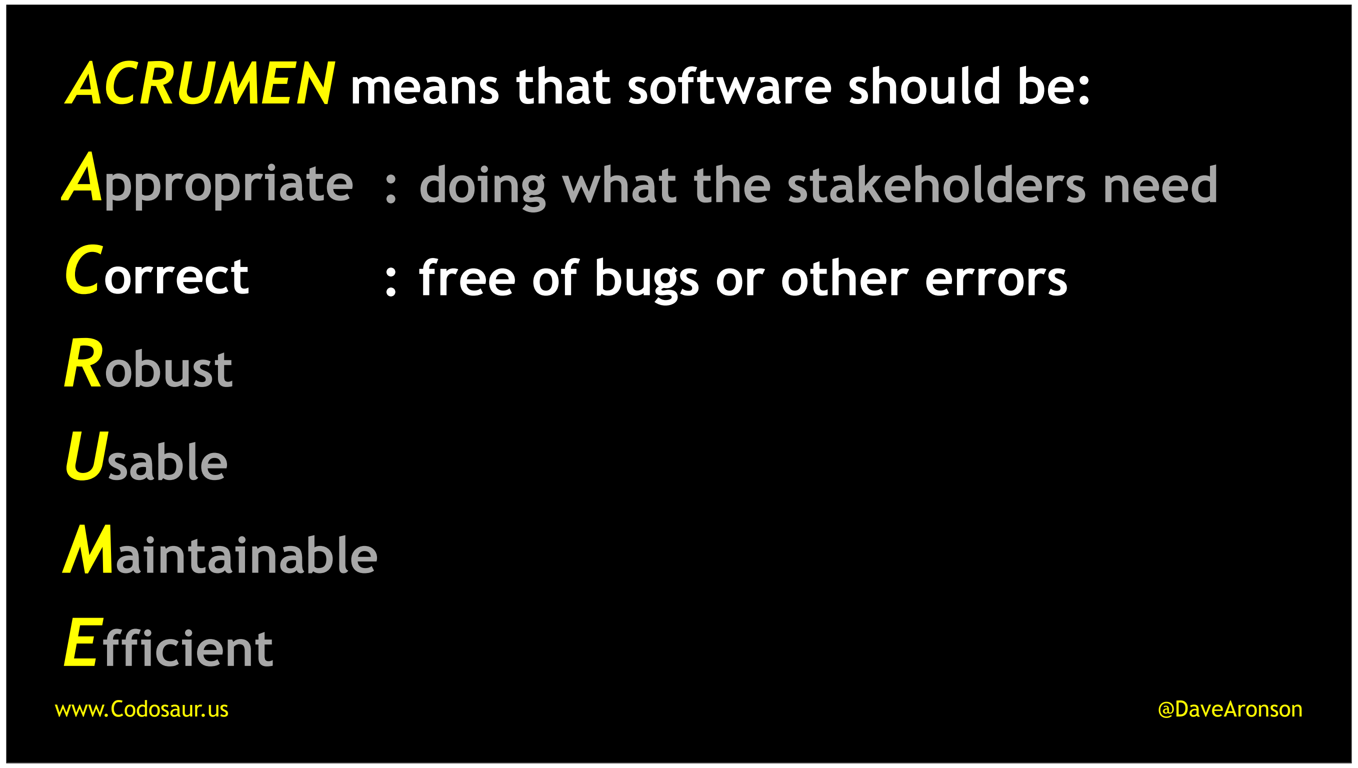
*M*aintainable

*E*fficient

www.Codosaur.us

@DaveAronson

. . . *doing what the stakeholders need* it to do. In other words, doing the *right job*, or more likely the right job***s***. And notice that I say "stakeholders", not "users"! The Project Management Institute defines "stakeholders" as everybody involved in, or affected by, the project, or in this case, the software. So yes, that *includes* the users, but also the people involved in, for instance, the development (hey, that's us!), the deployment, the monitoring, the customer service, the sales, the marketing, and so on, *and* often the *management* of many of those people. Then, once those right jobs have been identified, it needs to be . . .



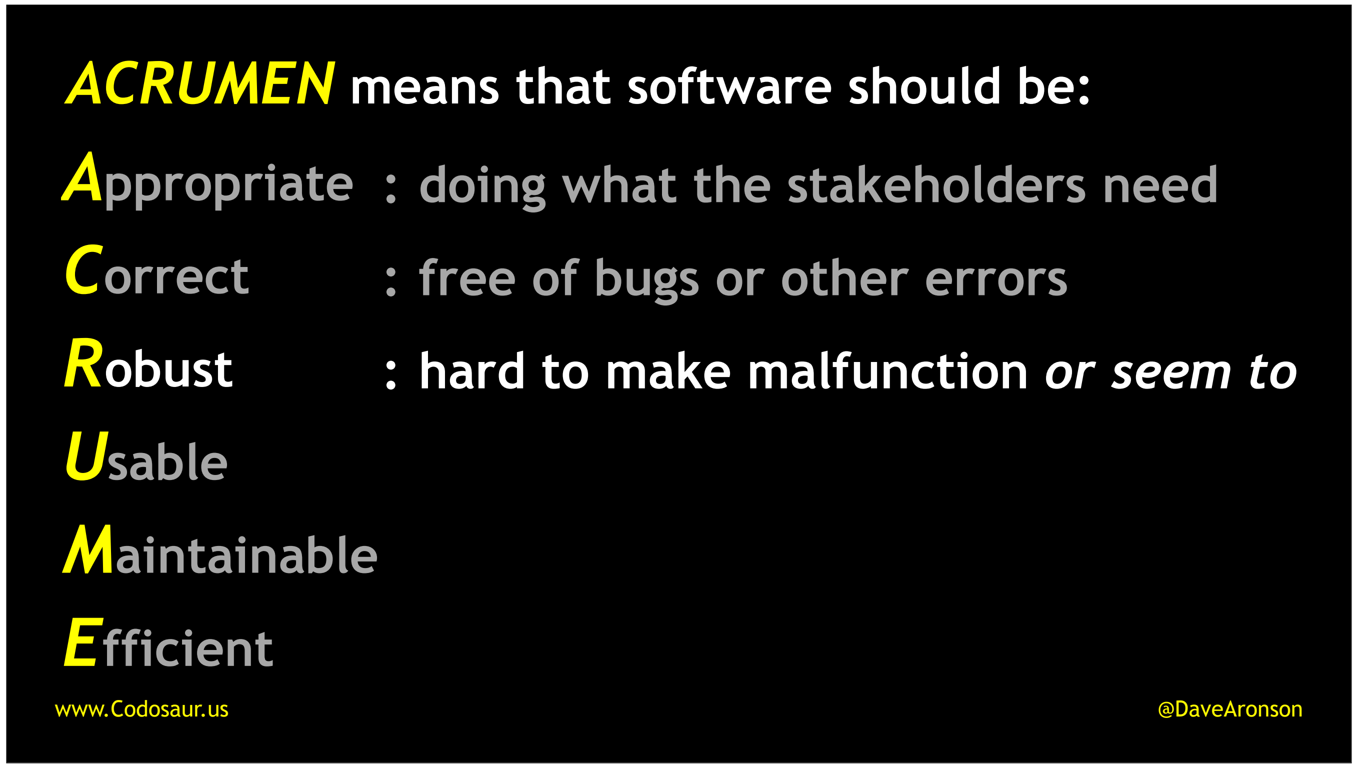
ACRUMEN means that software should be:

Appropriate	: doing what the stakeholders need
Correct	: free of bugs or other errors
Robust	
Usable	
Maintainable	
Efficient	

www.Codosaur.us

@DaveAronson

. . . *doing* those jobs *without bugs or other errors,* or in other words, doing the right jobs *right*. This one is pretty much exactly what it says on the tin, so, moving on, it should be . . .



ACRUMEN means that software should be:

*A*ppropriate : doing what the stakeholders need

*C*orrect : free of bugs or other errors

*R*obust : hard to make malfunction *or seem to*

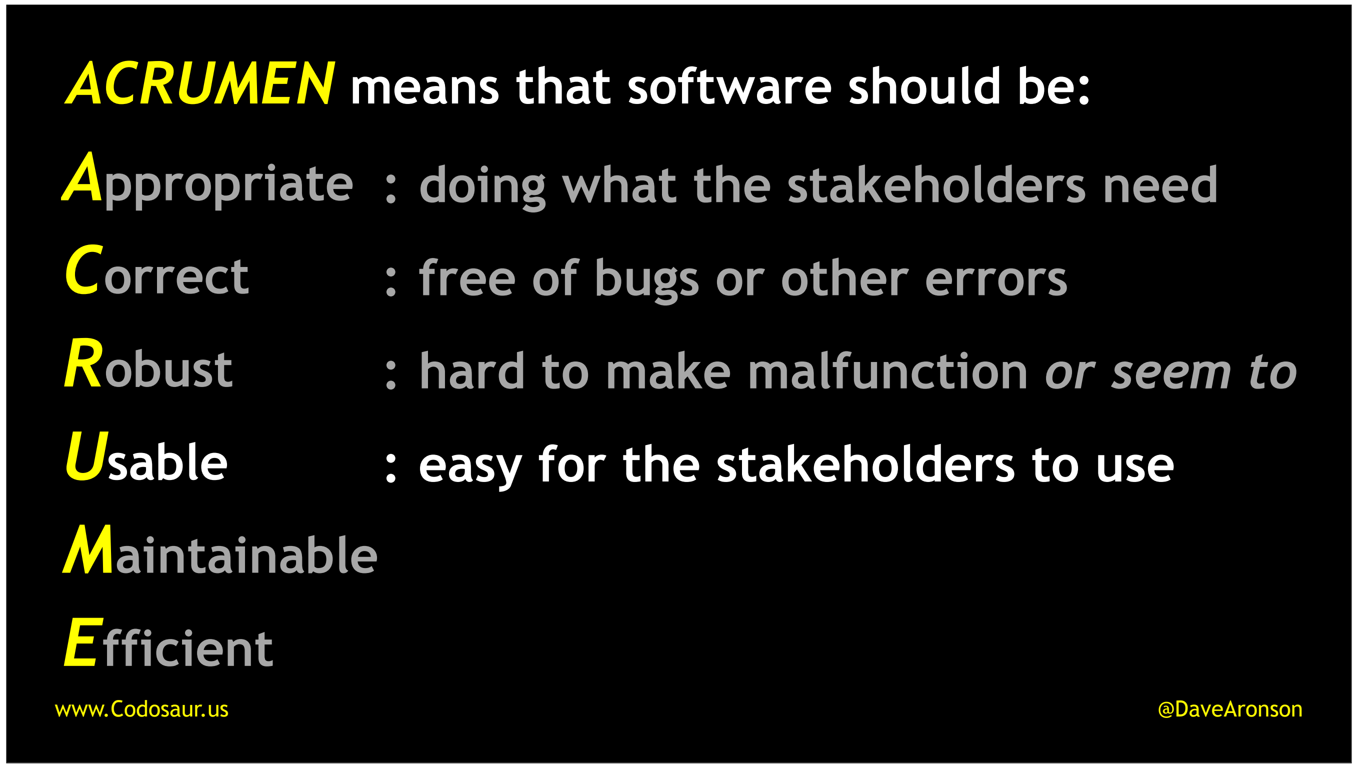
*U*sable

*M*aintainable

*E*fficient

www.Codosaur.us @DaveAronson

. . . hard for anyone to make it crash, or otherwise malfunction, or even seem to, whether deliberately or accidentally. On the other claw, it should be . . .



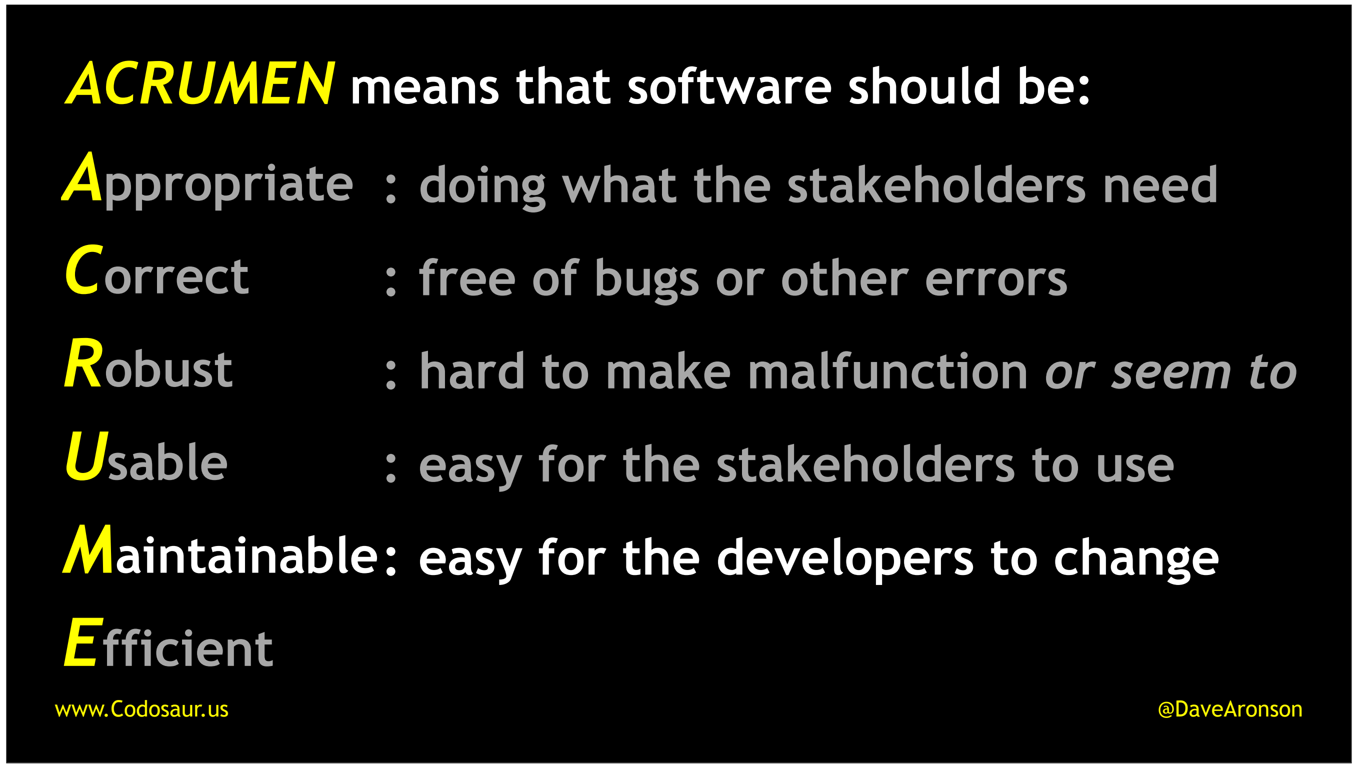
ACRUMEN means that software should be:

***A**ppropriate*	: doing what the stakeholders need
***C**orrect*	: free of bugs or other errors
***R**obust*	: hard to make malfunction *or seem to*
***U**sable*	: easy for the stakeholders to use
***M**aintainable*	
***E**fficient*	

www.Codosaur.us

@DaveAronson

. . . easy for the stakeholders to use, but here, “use” doesn’t necessarily mean that they’re interacting directly with the software. They could just be using some *data* it produces, whether as part of its main functionality, or its logs, or whatever. It should also be easy . . .



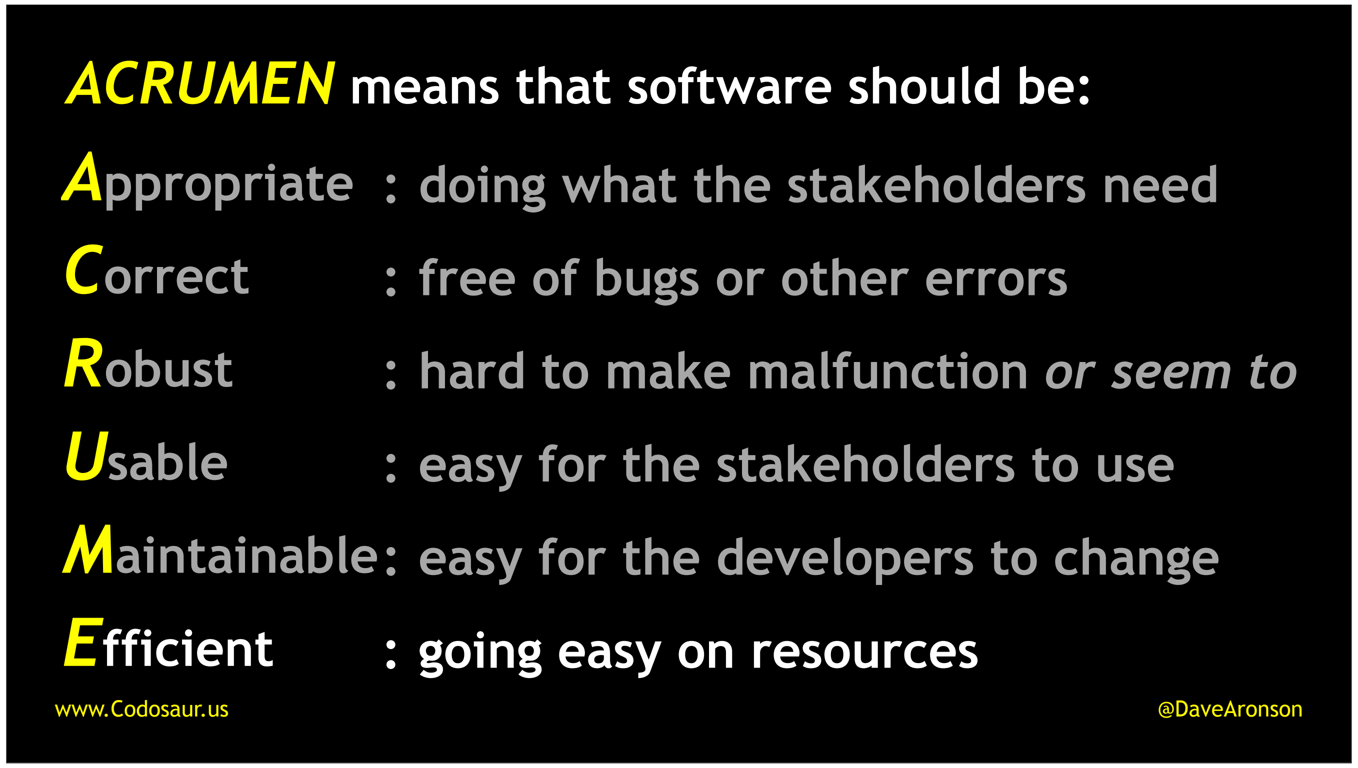
ACRUMEN means that software should be:

*A*ppropriate	: doing what the stakeholders need
*C*orrect	: free of bugs or other errors
*R*obust	: hard to make malfunction *or seem to*
*U*sable	: easy for the stakeholders to use
*M*aintainable	: easy for the developers to change
*E*fficient	

www.Codosaur.us

@DaveAronson

. . . for the developers to change. One might consider that a special case of Usability, but I think it's important enough, especially to this crowd, to stand on its own. Other than that, it's pretty much what it sounds like. And last, *dead last* despite how we developers tended to absolutely worship this just a few short decades ago, it should . . .



ACRUMEN means that software should be:

*A*ppropriate	: doing what the stakeholders need
*C*orrect	: free of bugs or other errors
*R*obust	: hard to make malfunction *or seem to*
*U*sable	: easy for the stakeholders to use
*M*aintainable	: easy for the developers to change
*E*fficient	: going easy on resources

www.Codosaur.us @DaveAronson

. . . go easy on resources, not only the technical ones that we usually think of, but *other* kinds as well.

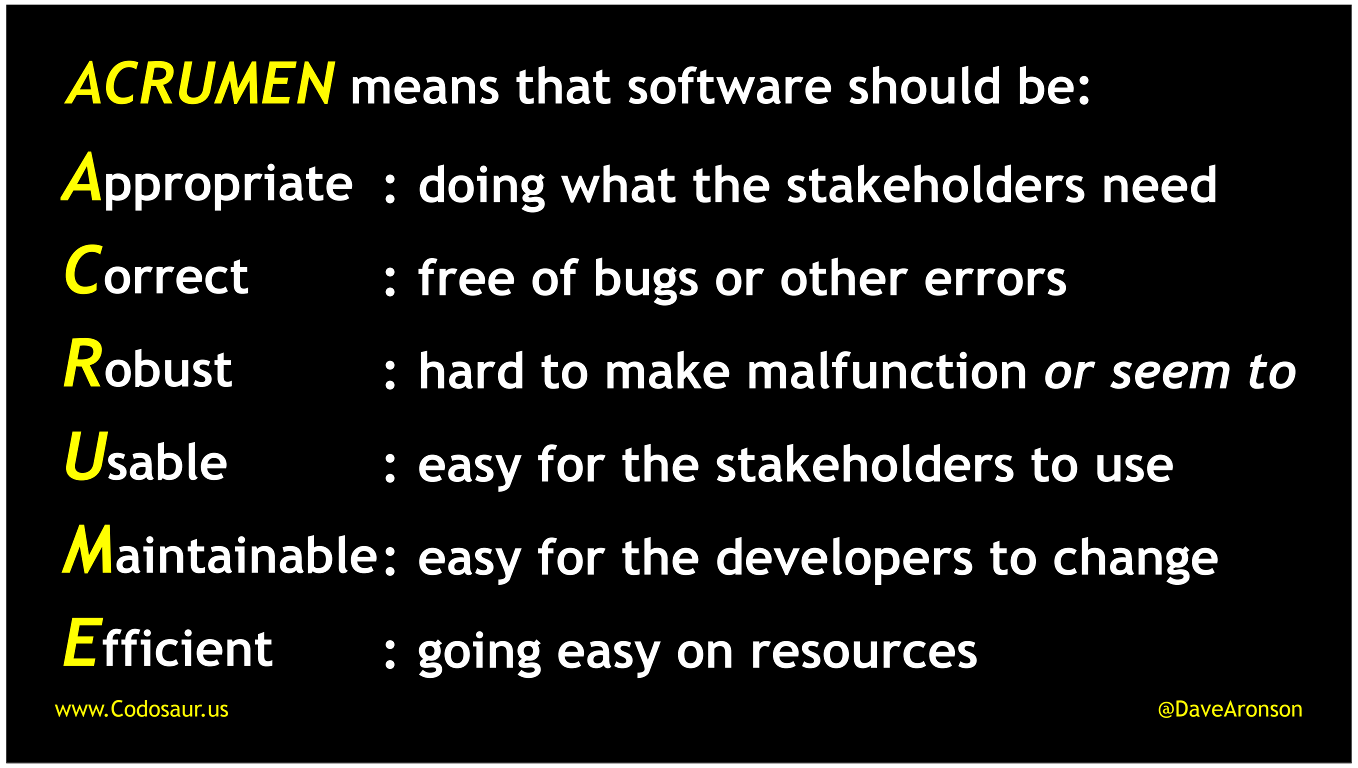
Now, I've said that ACRUMEN consists of six aspects, and you see six listed up there, but . . . it has seven letters! So what does the N stand for? Nnnnn . . .

www.Codosaur.us

@DaveAronson

. . . nothing! I just tacked it on to make a real word, even if an obsolete one.

Now, while . . .



ACRUMEN means that software should be:

*A*ppropriate	: doing what the stakeholders need
*C*orrect	: free of bugs or other errors
*R*obust	: hard to make malfunction *or seem to*
*U*sable	: easy for the stakeholders to use
*M*aintainable	: easy for the developers to change
*E*fficient	: going easy on resources

www.Codosaur.us @DaveAronson

. . . the basic definition is fresh in our minds, I'll address some frequently asked questions. You might want to take a picture of either this slide, or the final one, which has the URL for the whole deck.

First, aside from going into detail on the tips how to achieve each aspect, how do we actually *use* ACRUMEN itself, the list of aspects?

Mainly, we can keep it in mind as a sort of . . .

CHECKLIST
www.Codosaur.us
Image: https://pixabay.com/en/checklist-list-check-check-list-911840/
@DaveAronson

. . . *checklist*, when writing or evaluating software. We can ask, *is* it Appropriate, *is* it Correct, and so on, or *how* good is it in each aspect, whether that be on some scale like 1 to 10 or whatever, or by simple triage, or is it *good enough* for *our needs?* And if the answer is ever that it's not good enough, we can ask, what can be done to *. . .*

www.Codosaur.us

Image: https://www.flickr.com/photos/syume/4908943709/

@DaveAronson

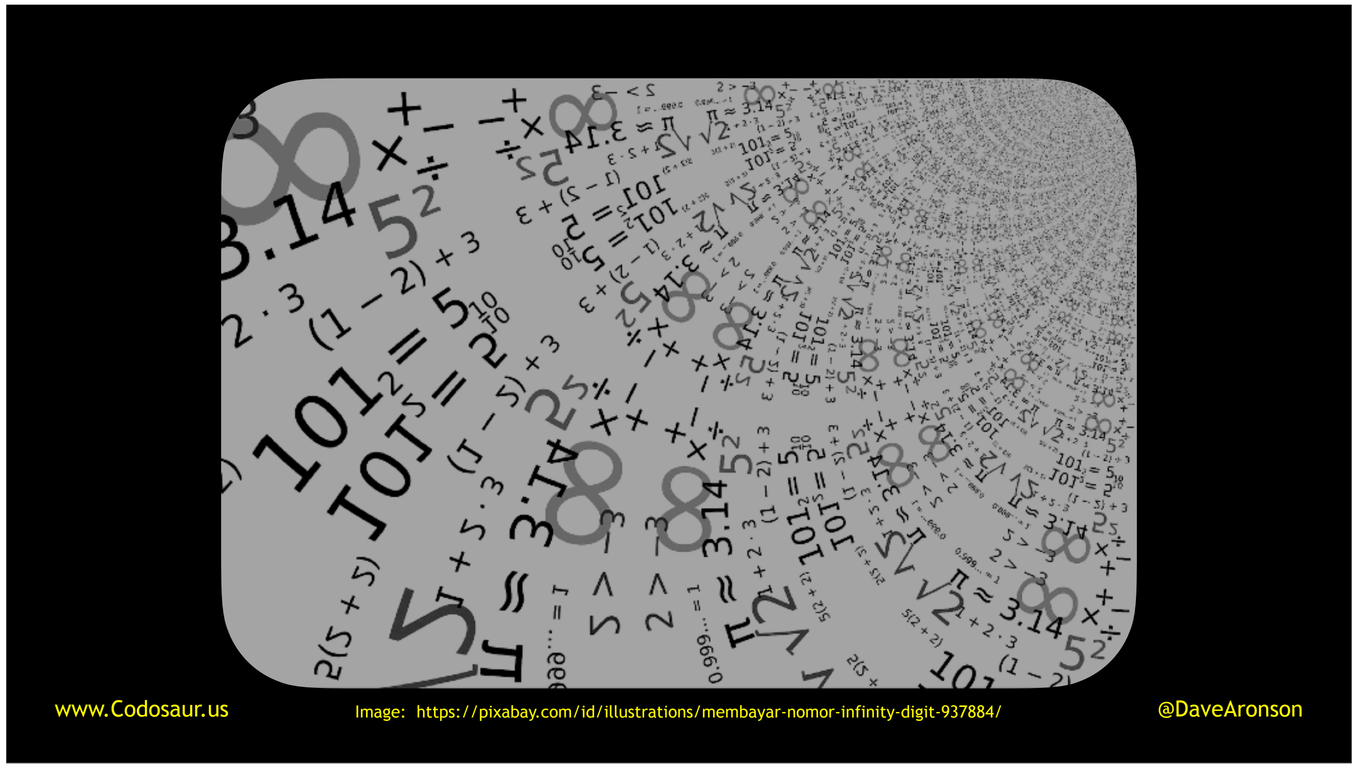
. . . *make* it so?

In the more immediate term, we can ensure that our current *projects* are likely to *meet* these criteria. In the longer term, we can ensure that our *processes support* these criteria, by including various helpful activities and requirements, and maybe even an explicit evaluation against the ACRUMEN aspects. We can also set . . .

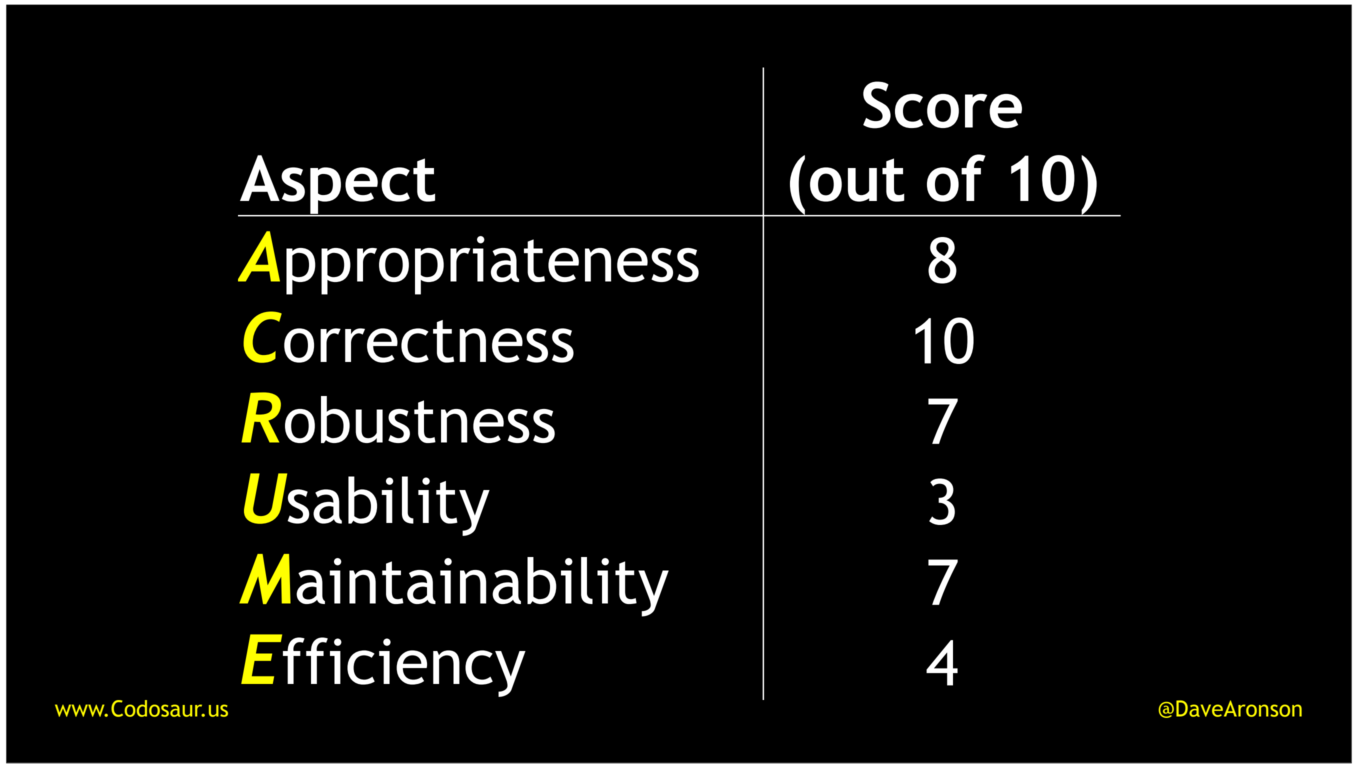
www.Codosaur.us
Image: https://www.flickr.com/photos/bensutherland/205606714
@DaveAronson

. . . targets, for how good we *need* some system to be in each aspect.

That leads us straight to the third most frequently asked question (after which, don’t worry, we'll backtrack to #2): . . .


www.Codosaur.us
Image: https://pixabay.com/id/illustrations/membayar-nomor-infinity-digit-937884/
@DaveAronson

. . . how can we quantify this, and boil it down to *one number* that shows the quality of a piece of software? Mainly I advise that you *don't* do that! Instead, at the very least, . . .



Aspect	Score (out of 10)
Appropriateness	8
Correctness	10
Robustness	7
Usability	3
Maintainability	7
Efficiency	4

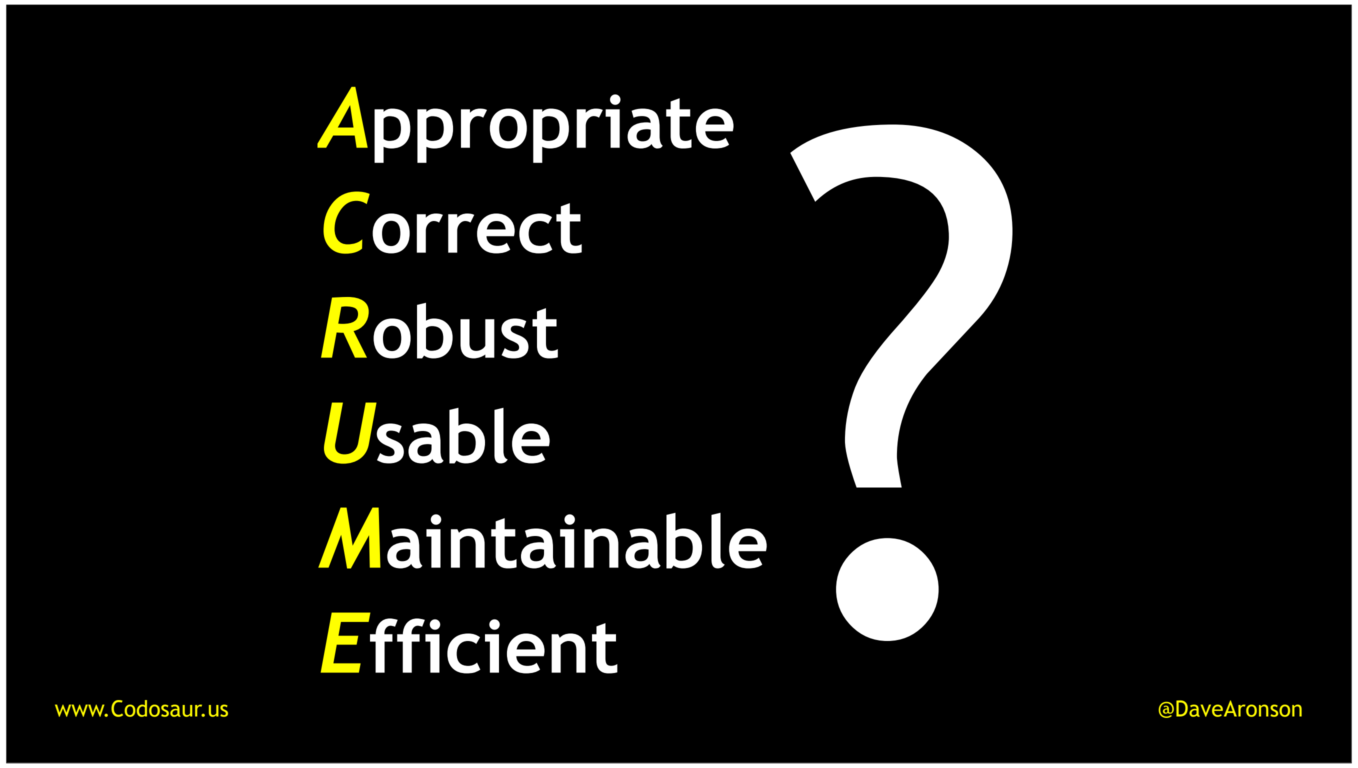
www.Codosaur.us

@DaveAronson

. . . keep *six* numbers, one for each aspect. Otherwise, you lose too much valuable information. A single number *might* tell you that the software is good or bad, maybe *how* good or bad it is, but a set of *six* numbers will tell you *how it is* good or bad.

For instance, with a chart like this, we're probably talking about a program that does most of what's needed, with minimal extra bells and whistles, does it absolutely correctly, but not very efficiently, I would guess it's doing it slowly, which is also impacting the usability. It's also fairly robust and maintainable, but could still use some improvement there too. These numbers can help *prioritize* further work on it. You don't have to get 10s across the board, but remember what I said about setting *targets*.

If you *really really must* put a single number on it, I would recommend you decide first how important each aspect is *for your use-case*, rate them on some scale, and use that to come up with a weighted average. And that leads us naturally back to the *second* most frequently asked question . . .



Appropriate
Correct
Robust
Usable
Maintainable
Efficient

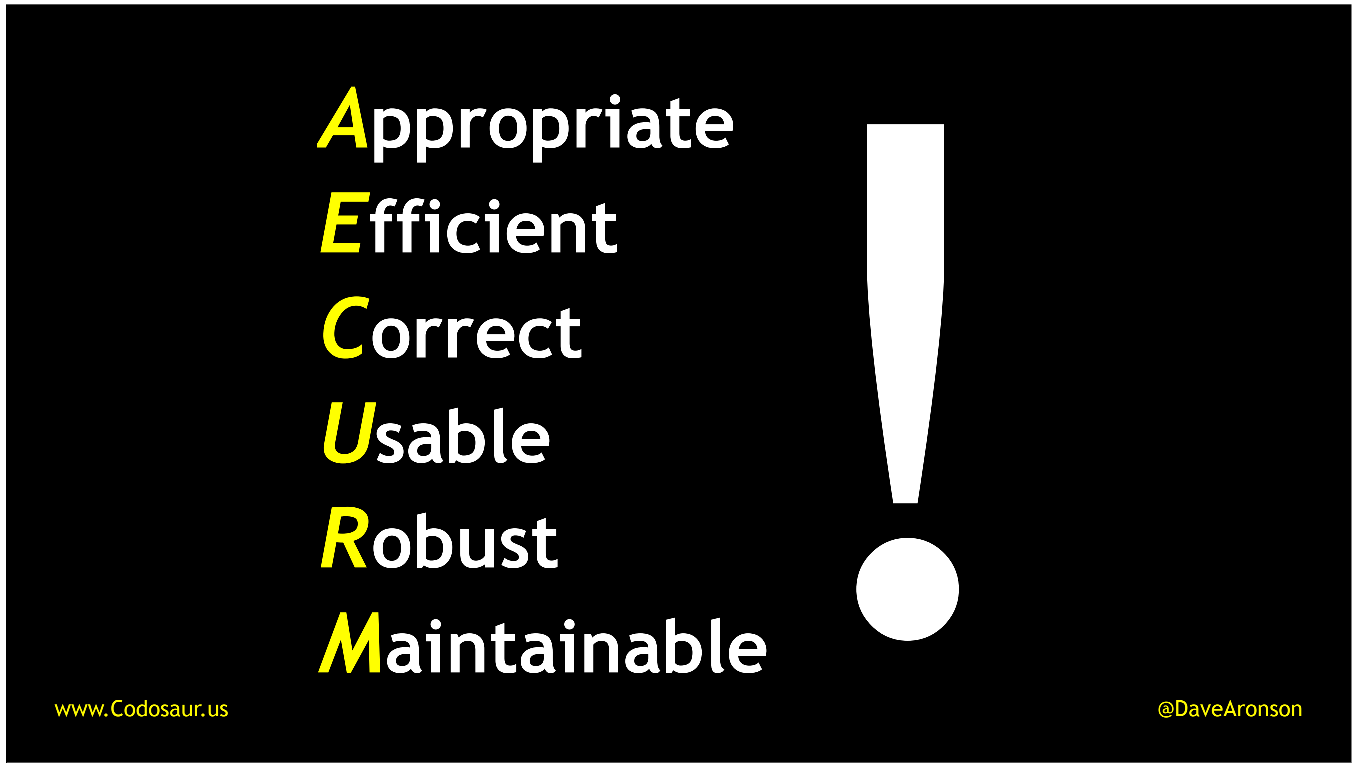
?

www.Codosaur.us

@DaveAronson

. . . is ACRUMEN, or rather ACRUME, always the right ordering? Some projects seems a little different.

As I've already implied a few times, the answer is, no, ACRUMEN is just the *typical* case. Your mileage (or outside the USA, maybe your kilometrage) may well vary. Consider, for instance, a company-internal command-line physics simulation tool, using a standard algorithm that will never change, finding a close-enough answer to something where precise calculation would take forever, like maybe weather forecasting. It absolutely needs to do the right job, but doesn't have to be precisely correct, just close enough. It might not need to be so robust because of the limited interfaces and fewer things to go wrong. Nor so usable because it's just for ourselves, not paying customers. Nor so maintainable, because the core logic is never going to change. But given such a complex domain, it *may* need to be very efficient in getting *to* that approximate answer. So, its prioritized list of aspects may well look more like . . .



Appropriate
Efficient
Correct
Usable
Robust
Maintainable
!

www.Codosaur.us

@DaveAronson

. . . this, AECURM, rather than ACRUME. The only real constant is that Appropriate will always be at the top. We'll see why very soon, because now we're going to look at each aspect in more detail, and up first is of course . . .

THE FOLLOWING PREVIEW HAS BEEN APPROVED FOR
APPROPRIATE AUDIENCES
BY THE MOTION PICTURE ASSOCIATION OF AMERICA, INC.

THE FILM ADVERTISED HAS BEEN RATED
G GENERAL AUDIENCES
All Ages

www.filmratings.com www.mpaa.org

www.Codosaur.us @DaveAronson

. . . Appropriateness.

If our software doesn't have this, then Nothing. Else. Matters. (PAUSE!) If our software is doing the *wrong job*, then it *doesn't matter* how *well* it's doing *the wrong job!* So, appropriateness is not only more important than any other aspect, it's even more important than . . .

CRUME
A
www.Codosaur.us
Image: https://pixabay.com/en/balance-scale-justice-law-judge-154516/ + my text & lines
@DaveAronson

. . . *all* of the others *put together*. (If you're trying to put one number on the quality of some software, you might want to remember that when you rate the importance of each aspect. Your rating for Appropriateness should be *at least* the sum of all the other aspects' ratings.) And yet, we developers are generally not taught that this is even "a thing", let alone one that *we* need to think about.

To *prove* the importance of being . . .

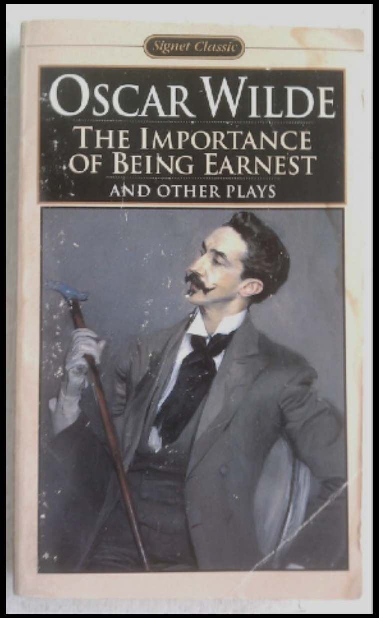


Image: https://www.flickr.com/photos/carlmikoy/6190241351

. . . Earnest, er, I mean, . . .

THE FOLLOWING PREVIEW HAS BEEN APPROVED FOR
APPROPRIATE AUDIENCES
BY THE MOTION PICTURE ASSOCIATION OF AMERICA, INC.

THE FILM ADVERTISED HAS BEEN RATED

G GENERAL AUDIENCES
All Ages

www.filmratings.com www.mpaa.org

www.Codosaur.us @DaveAronson

. . . Appropriate, let's try a little thought experiment. Suppose you want a program to play . . .

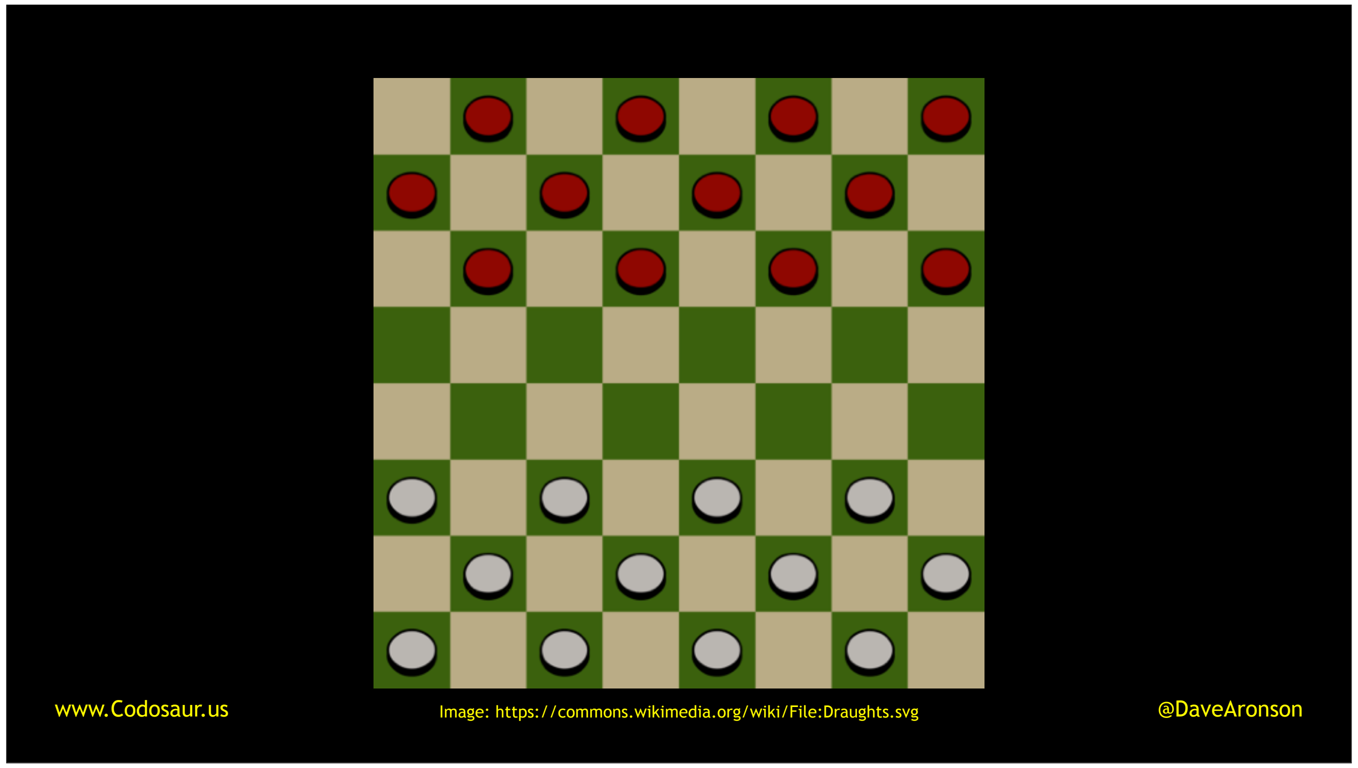
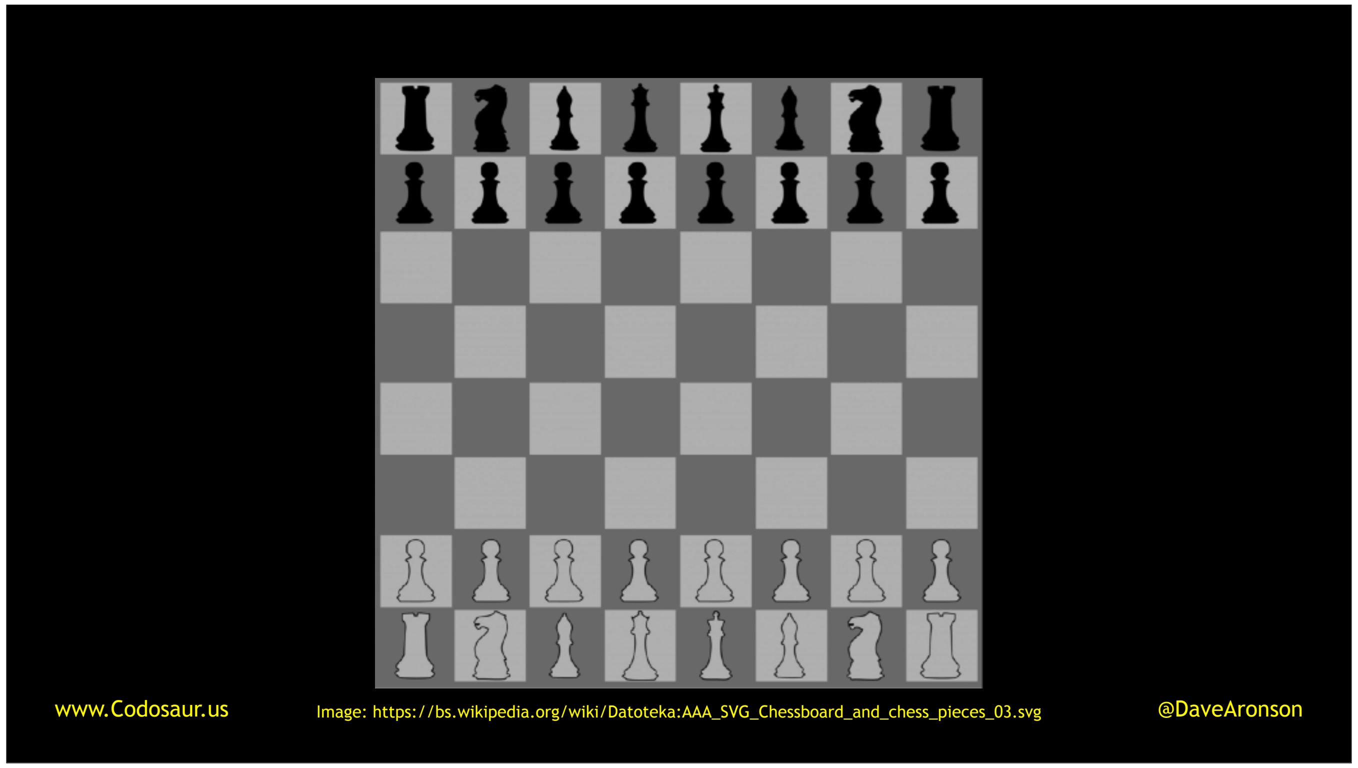

www.Codosaur.us

Image: https://commons.wikimedia.org/wiki/File:Draughts.svg

@DaveAronson

. . . *checkers* (or as you call it in British English, draughts), and I write for you the world's greatest . . .



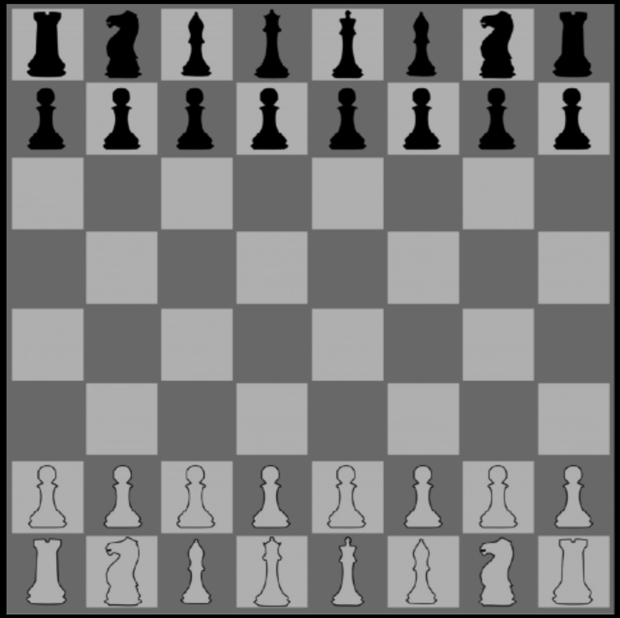
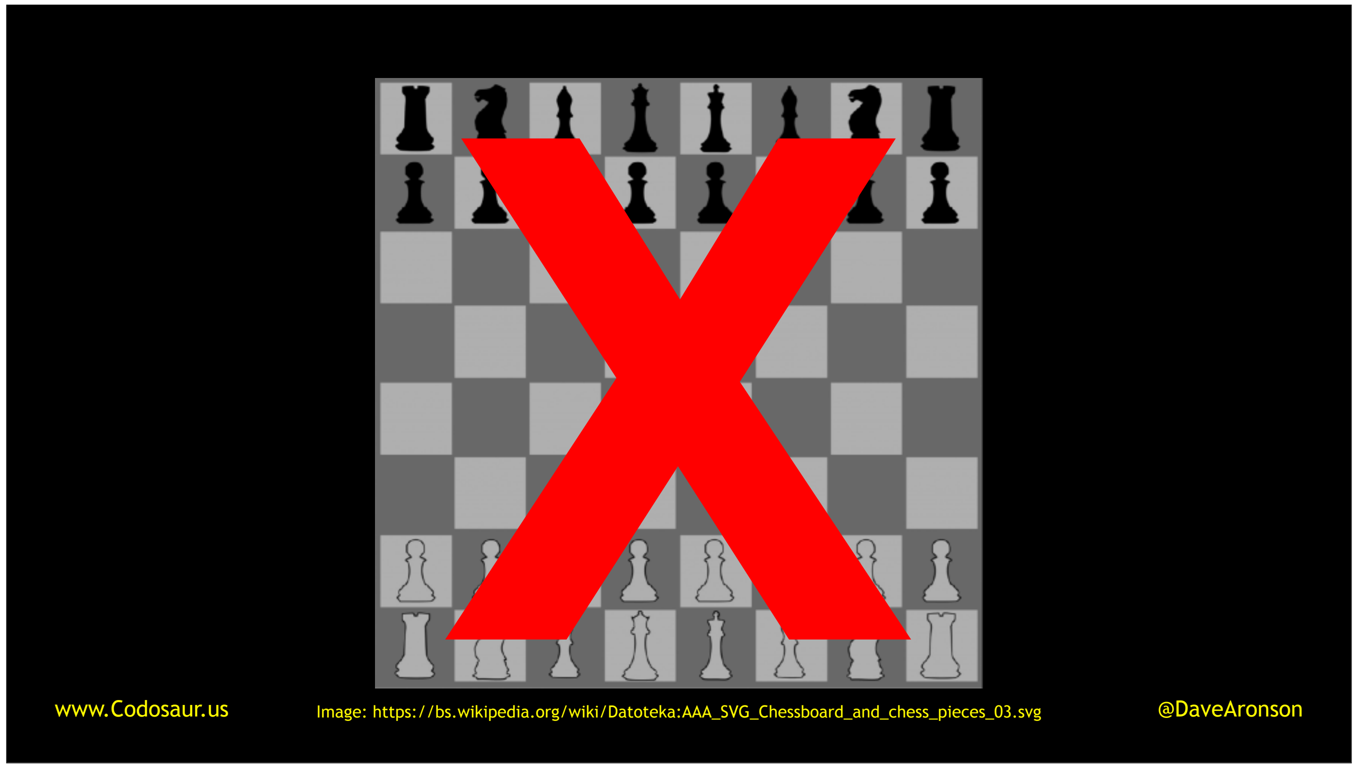


Image: https://bs.wikipedia.org/wiki/Datoteka:AAA_SVG_Chessboard_and_chess_pieces_03.svg

. . . *chess* playing program. It's as correct, robust, usable, maintainable, and efficient as anyone could ever want. But are you going to be happy with it? (PAUSE FOR REACTIONS, AND COMMENT ON THEM) Probably . . .


www.Codosaur.us
Image: https://bs.wikipedia.org/wiki/Datoteka:AAA_SVG_Chessboard_and_chess_pieces_03.svg
@DaveAronson

. . . not. But why not, if it's such a great program? (PAUSE!) Because . . . it's not checkers. (Or draughts, whatever.) It's not what you asked for. It's not what you (presumably) need. Or in ACRUMEN terms, it's not *appropriate*.

So now that we know how important this is, how do we achieve it? In an ideal world, we would have . . .

www.Codosaur.us
Image: https://www.flickr.com/photos/wocintechchat/22543243101
@DaveAronson

. . . frequent direct contact with the stakeholders — of *all* kinds! Ideally it would be face to face, or as close to that as practical. We can ask what they want, and break it down into smaller and smaller pieces. Developers should be good at that, it's basically how programming works under the hood, er, I mean, bonnet! But . . . we should go a step further, and ask WHY they want each thing. This will help reveal what they really NEED, which is what *we* really need to satisfy, as opposed to what they say they want, which makes it two steps removed from there.

Unfortunately, we don't usually get that opportunity. Second best is to bring in the experts, which in this case would be Requirements Analysts. But on this . . .

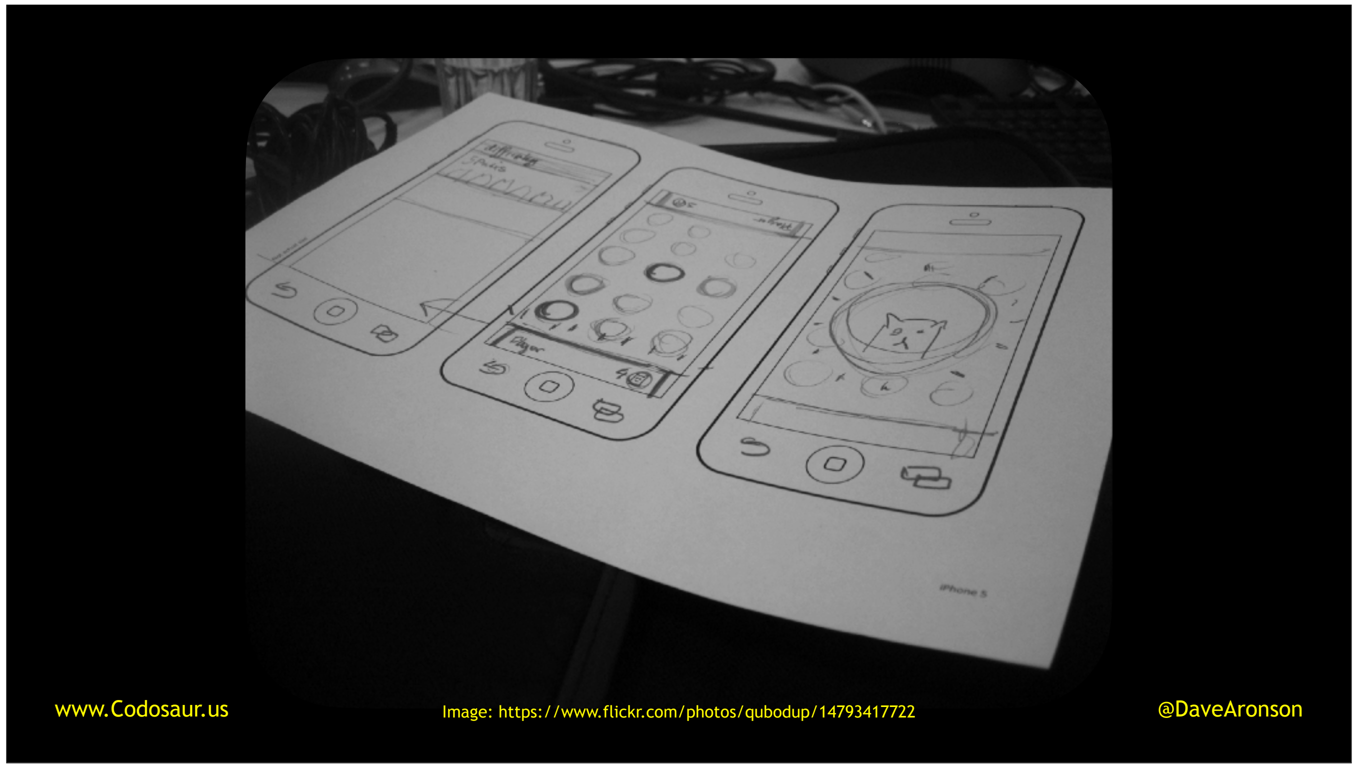
www.Codosaur.us

Image: https://roundupreads.jsc.nasa.gov/pages.ashx/699/10%20things%20to%20know%20and%20share%20about%20the%20Eclipse%20Across%20America

@DaveAronson

. . . planet, we usually don’t get those either, at least outside of huge companies. We might not even have *business* analysts available, the next step down. So we usually have to settle for occasional remote indirect contact with a representative of some stakeholders, such as a Product Owner in Scrum. It doesn't work quite as well, but having *some* communication with *someone* with *some* clue, is *vital*.

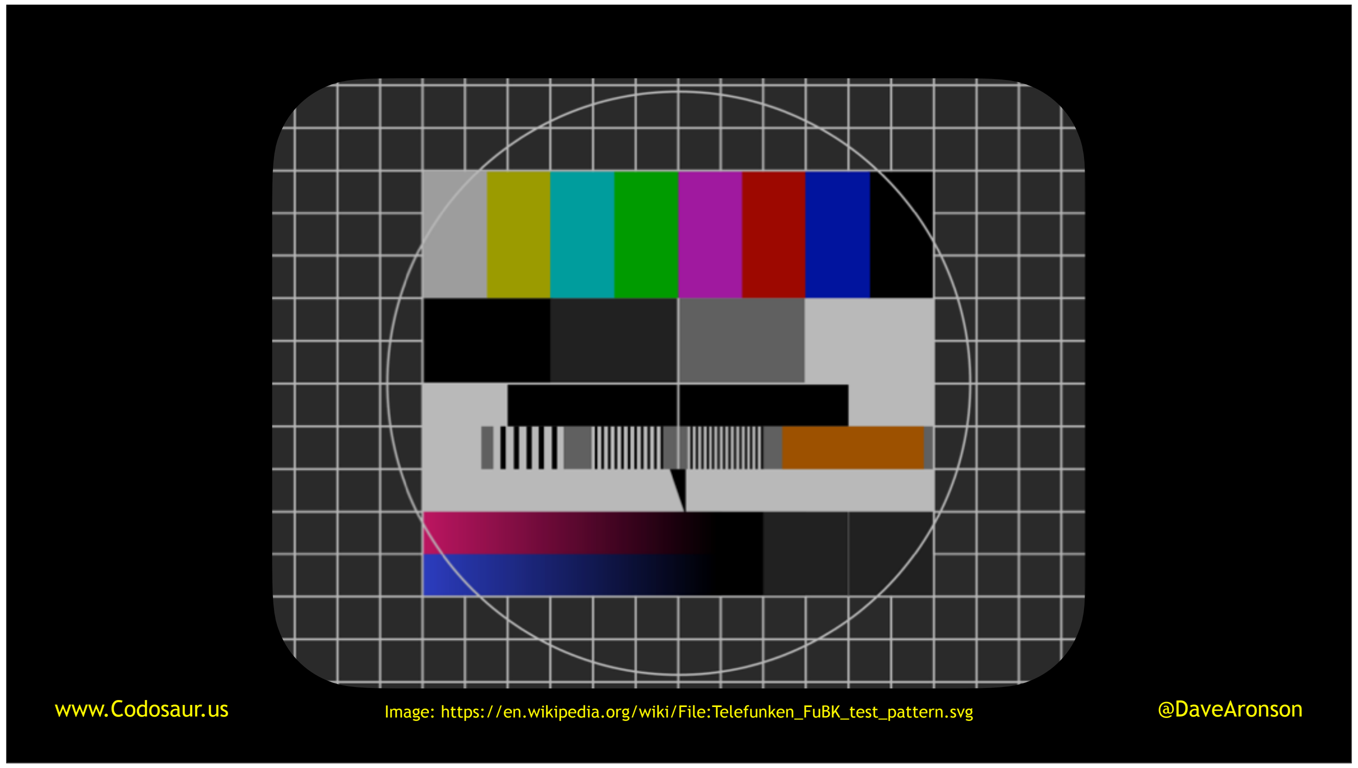
Once we think we have a good grasp of their needs, we can show them . . .


www.Codosaur.us Image: https://www.flickr.com/photos/qubodup/14793417722 @DaveAronson

. . . mockups and prototypes of what we *intend* to do, and demos of what we have *already* done. This gives them a chance to *correct our wrong ideas* of their needs, before we go too far down the wrong rabbit-hole. Has anyone else been there, wasting time implementing the *wrong thing?* (PAUSE FOR REACTIONS) Looks like a lot of you, though fewer than I'd have thought. The rest of you, you probably did, but might not have known it at the time.

Anyway, ideally, show them these things *frequently,* as a sort of continuous course correction. Frequent feedback *from* the stakeholders is even *more* important than being able to ask them questions. This is an example of a concept that's gotten much wider recognition over the past couple decades or so: tight feedback loops.

There's another thing, though, that I'll be returning to over and over in this talk, in various forms. We can propose . . .


www.Codosaur.us
Image: https://en.wikipedia.org/wiki/File:Telefunken_FuBK_test_pattern.svg
@DaveAronson

. . . *tests!* In particular, I recommend the Given/When/Then pattern:
given: these preconditions, such as data being in a certain state;
when: this happens, usually some kind of input from users, or a timer, or a sensor, or another system, that is meant to result in some *known operation* in our system;
then: this is the result, usually either something the user sees on the screen, or data being in a desired new state.
This makes a great link between the worlds of business and tech, because the business people can understand it, and we can turn it into a runnable test.

Our next aspect is . . .

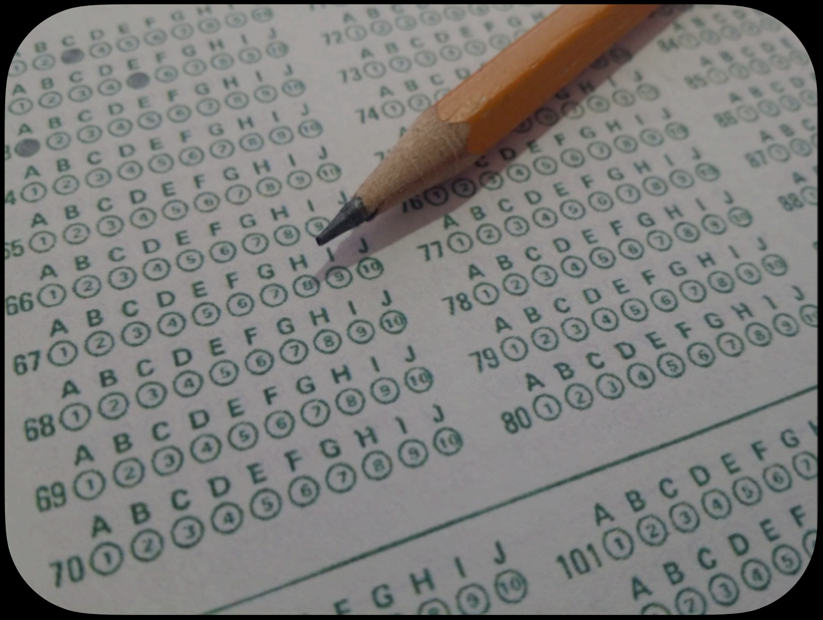
. . . correctness. If our software doesn't have this, then, it has bugs, and nobody likes that! Not only do the users not like *using* buggy software, we *developers* don't like having our *names* on it! Am I right? Does anybody here NOT take at least *that* much pride in their work? (PAUSE FOR REACTIONS)

Nothing can actually *stop* us from *writing* code that isn't correct, at least with the reasonably effective tools we have today, like our IDEs with their completions and AI assistants and so on, as opposed to the less effective drag-and-drop and similar systems. So, the big question is: . . .

www.Codosaur.us
Image: https://vectorportal.com/vector/knowledge-concept/31357
@DaveAronson

. . . how do we *know*? (PAUSE!)

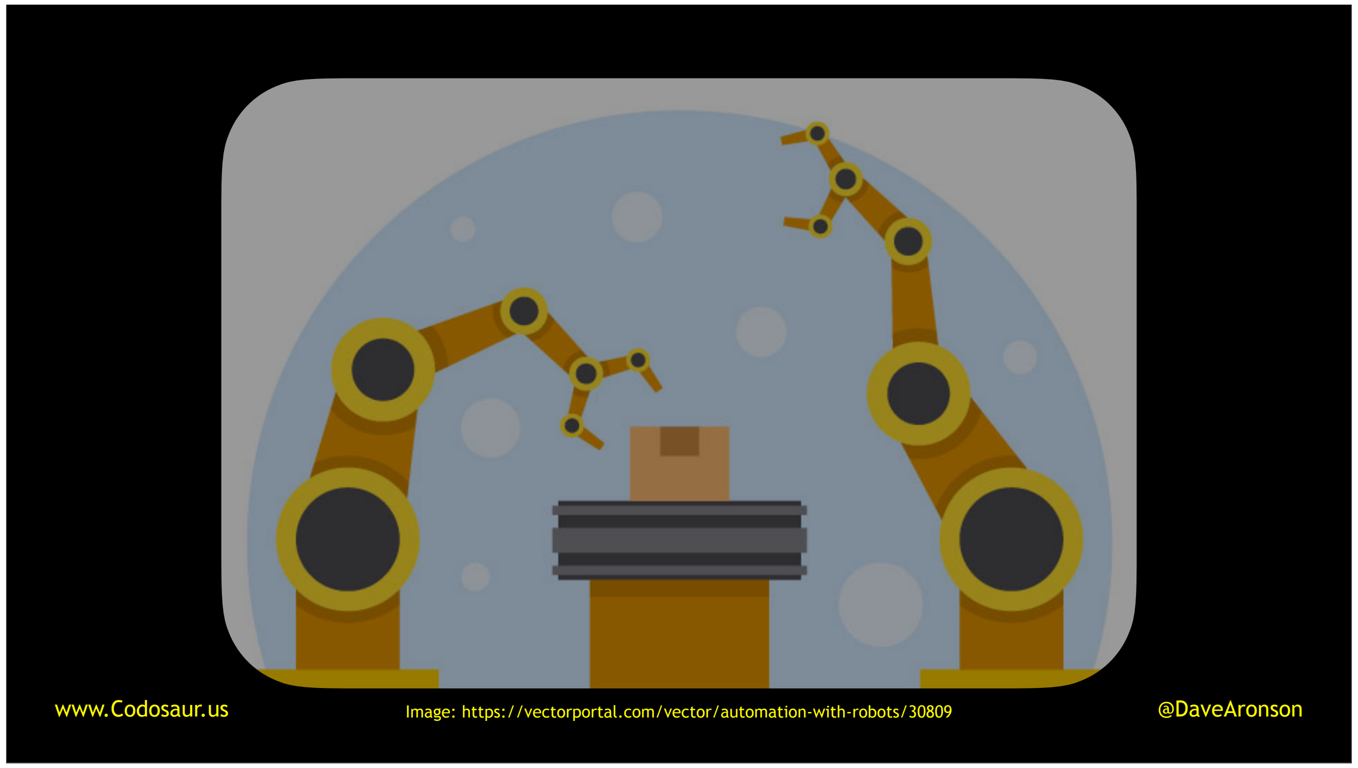
I just mentioned it, and a lot of you probably knew already, that . . .



www.Codosaur.us Image: https://pixabay.com/fi/photos/testi-testaus-kupla-muoto-986935/ @DaveAronson

. . . *tests* prove whether or not our code is correct . . . *assuming* of course, and keeping in mind what happens when you assume, that the tests *themselves* are correct. Actually, even then, it's not quite true, but I'll get to that in a moment.

Ideally our tests are . . .


www.Codosaur.us
Image: https://vectorportal.com/vector/automation-with-robots/30809
@DaveAronson

. . . automated, which makes them more repeatable, and maintainable than manual tests, and *much* faster, so they get us vital feedback while the problem at hand is still fresh in our minds (with those tight feedback loops I mentioned earlier) . . . again, *assuming* something about our tests, in this case that they are reasonably *efficient*.

We can start with the tests that the stakeholders approved for the sake of Appropriate-ness. These are likely to be . . .

www.Codosaur.us

Image: http://www.hill.af.mil/News/Article-Display/Article/838188/commentary-diamond-in-the-rough-commander-shares-514th-flts-mission/

@DaveAronson

. . . *high*-level types like end-to-end system tests, feature tests, and so on. But that's not all we need. As we implement the system, we should still add our own . . .

www.Codosaur.us Image: http://www.eglin.af.mil/News/Article/814410/marines-fly-first-f-35-stovl-mission-at-eglin/ @DaveAronson

. . . *low*-level tests, like unit tests, integration tests, and so on.

Even with all of that, though, normal tests like these can only prove the correctness of cases *we thought* to test. However, there are some advanced techniques that can help find edge cases, exceptions, and other unusual cases we didn't think of. For instance . . .

www.Codosaur.us

Image: https://www.flickr.com/photos/athomeinscottsdale/3279949186

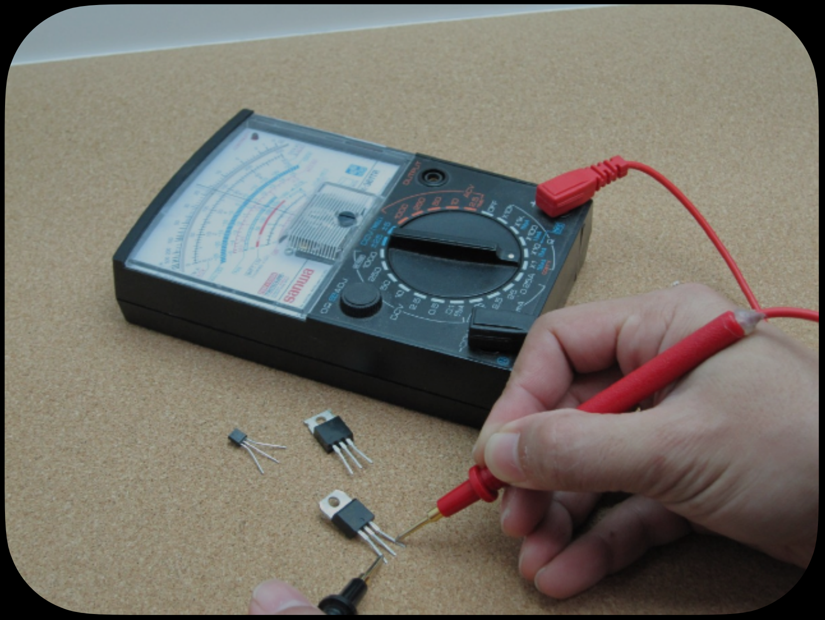
@DaveAronson

. . . property-based testing tests whether some desired property of the output, or its relationship to the input, what formal computer scientists might call an *invariant*, holds true for all valid inputs. A property testing tool makes up lots of random test data to try, somewhat like the security concept of "fuzzing", but staying *within* defined bounds of validity, rather than trying to find and exceed them to see what happens. If it finds a valid input that makes our property fail, that means that there is at least one case that we didn't consider, and quite likely an edge or a corner.

www.Codosaur.us
Image: https://pixabay.com/en/genetic-modification-mutant-mutation-549889/
@DaveAronson

Mutation testing runs our tests against slightly altered versions of our code, called “mutants”, usually containing one tiny change, or mutation, like changing a plus to a minus, or reversing the operands of a division. Each of these “mutants” should make at least one test fail. If not, it may be a false alarm, but it usually means one of two problems, or maybe even both. The most likely is that our test suite isn't *strict* enough to *catch* the difference the mutation made — no matter what our so-called “coverage” report might say. Or, it might mean that our code isn't *meaningful* enough for that mutation to make a *difference* in its *behavior*. For instance, some code may be redundant or unreachable, or at least its functionality is not worth testing, like debugging traces. BTW, I also speak on mutation testing at conferences. I won’t be doing that talk here this year, but you can find many of the past versions on Youtube.

Also, again we can bring in experts, which in this case would be . . .

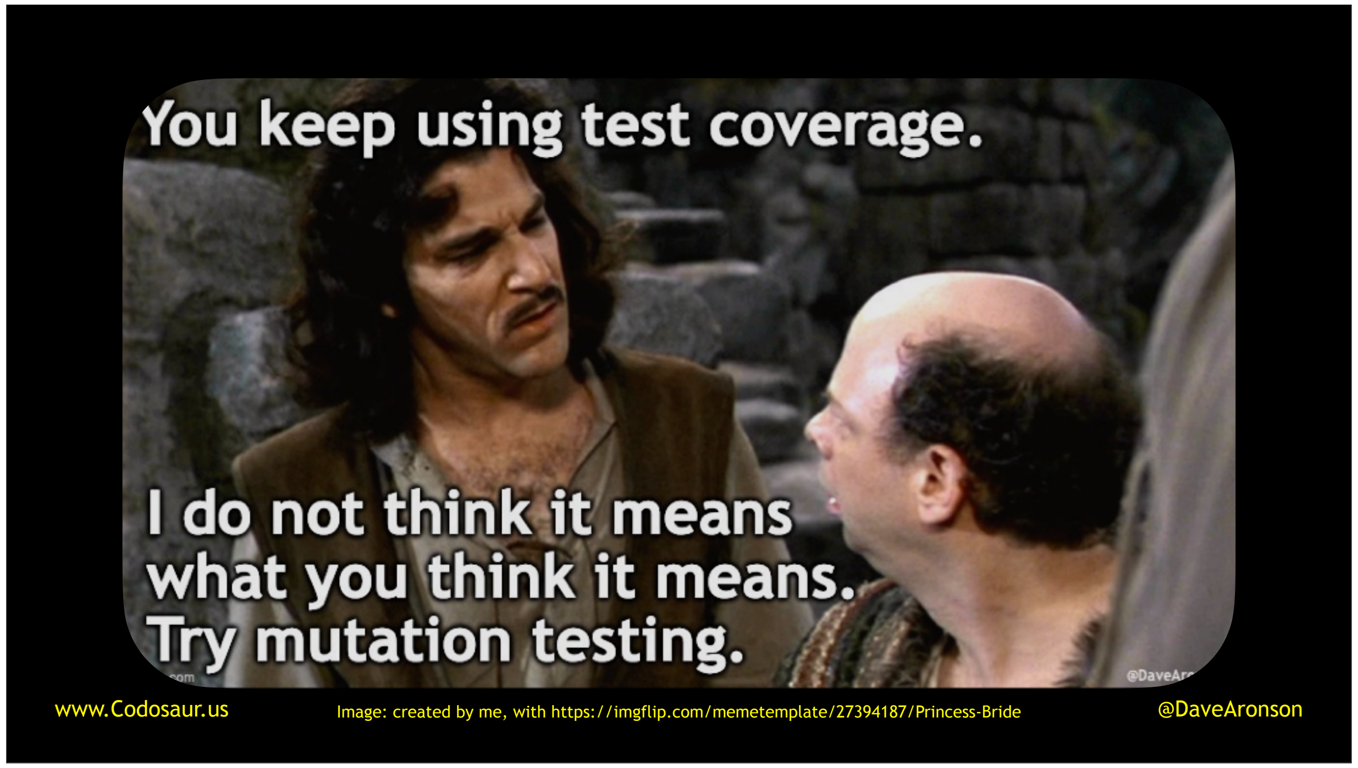


www.Codosaur.us Image: https://pxhere.com/en/photo/690933 @DaveAronson

. . . testers. Yes, we should do some fairly thorough testing ourselves, but we have extensive knowledge of how the system works. You might *think* that that would be a *good* thing, but in this case it . . .

. . . blinds us to many of the ways that users could *mis*use the system. Professional testers are very good at thoroughly probing a system for ways it can be misused, maybe even abused — though that in particular is a different domain, that we'll get to later.

Anyway, we should have enough test coverage, of assorted kinds and levels, *and verified to actually* . . .


You keep using test coverage.
I do not think it means what you think it means. Try mutation testing.
www.Codosaur.us
Image: created by me, with https://imgflip.com/memetemplate/27394187/Princess-Bride
@DaveAronson

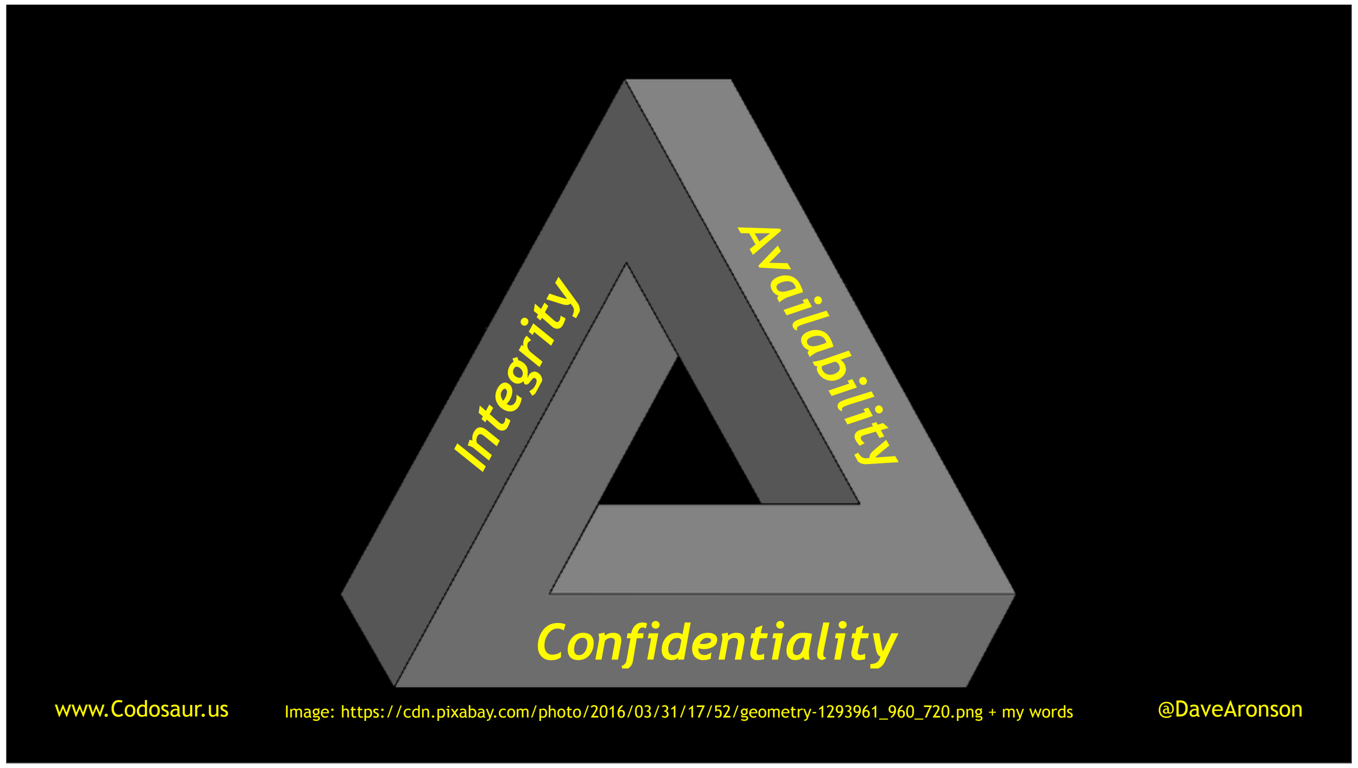
. . . *mean what we think it means* (PAUSE TO LET THEM READ SLIDE), through techniques such as mutation testing, for us to have strong confidence in the correctness of our code.

Next up we have . . .

ROBUST
901A
HANOMAG
www.Codosaur.us
Image: https://commons.wikimedia.org/wiki/File:Hanomag_Robust_901A_Cloppenburg.jpg
@DaveAronson

. . . robustness. If our software doesn't have this, then, *at best*, it may simply show a lot of error messages, and *seem* fragile and unreliable, or it may crash a lot and actually *be* fragile and unreliable, or it may even . . . Get Hacked, because Robustness includes Security.

The short explanation is that it's hard to make the software malfunction (or even seem to), but what does *that* mean?! There are a few other things, but most of what I mean is covered by a core concept of information security: . . .

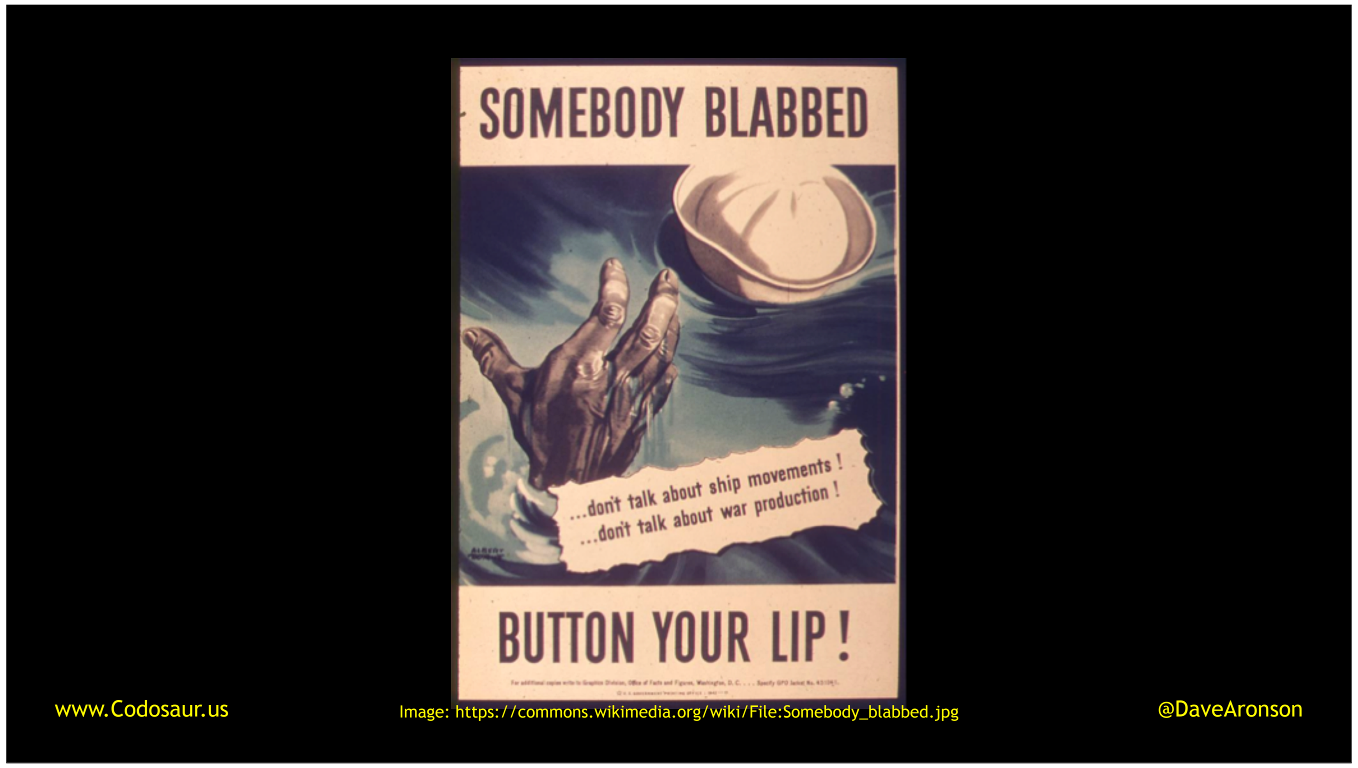


www.Codosaur.us

Image: https://cdn.pixabay.com/photo/2016/03/31/17/52/geometry-1293961_960_720.png + my words

@DaveAronson

. . . the CIA Triad. No, it's nothing to do with spies and gangsters, it's this triangle here, of Confidentiality, Integrity, and Availability. So, robust software does *NOT:*

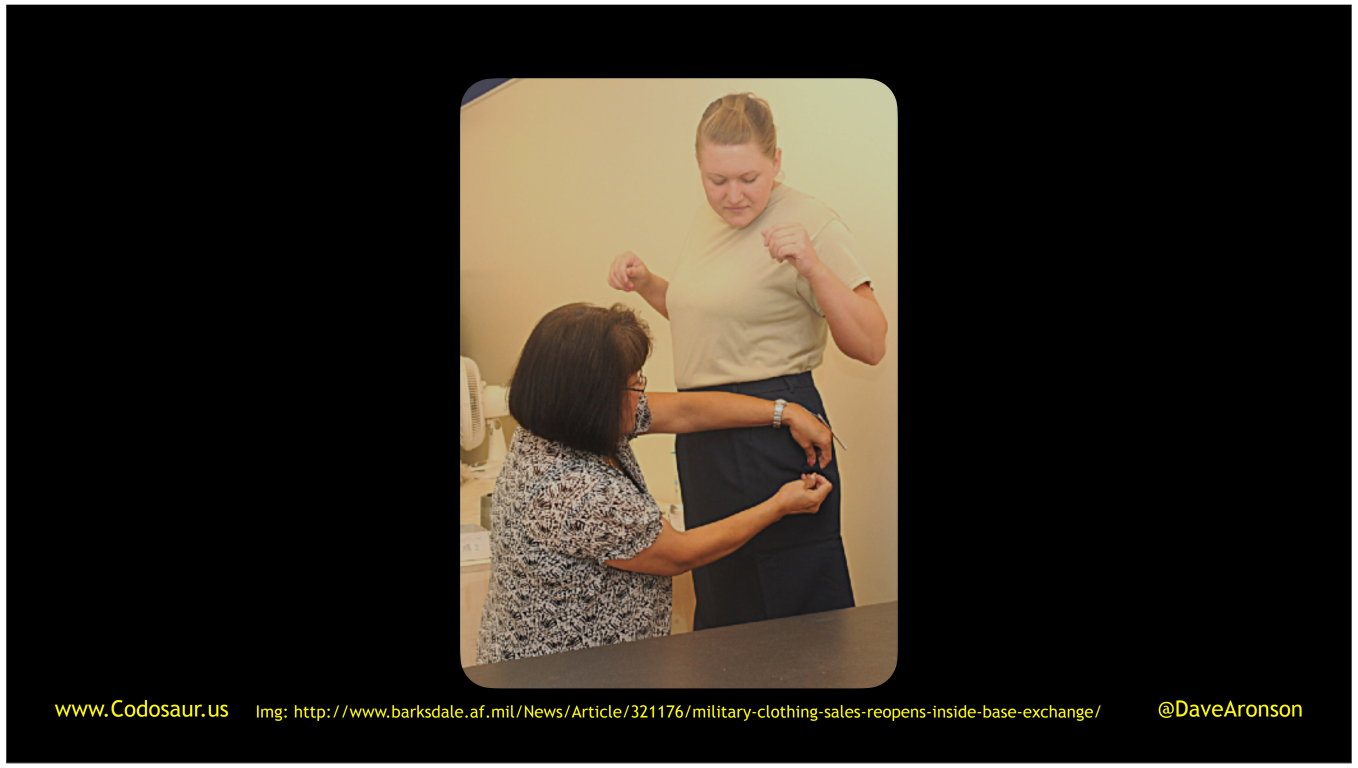

SOMEBODY BLABBED
...don't talk about ship movements!
...don't talk about war production!
BUTTON YOUR LIP!

www.Codosaur.us

Image: https://commons.wikimedia.org/wiki/File:Somebody_blabbed.jpg

@DaveAronson

. . . reveal data when it’s not supposed to, . . .



www.Codosaur.us Img: http://www.barksdale.af.mil/News/Article/321176/military-clothing-sales-reopens-inside-base-exchange/ @DaveAronson

. . . alter data when it’s not supposed to, . . .

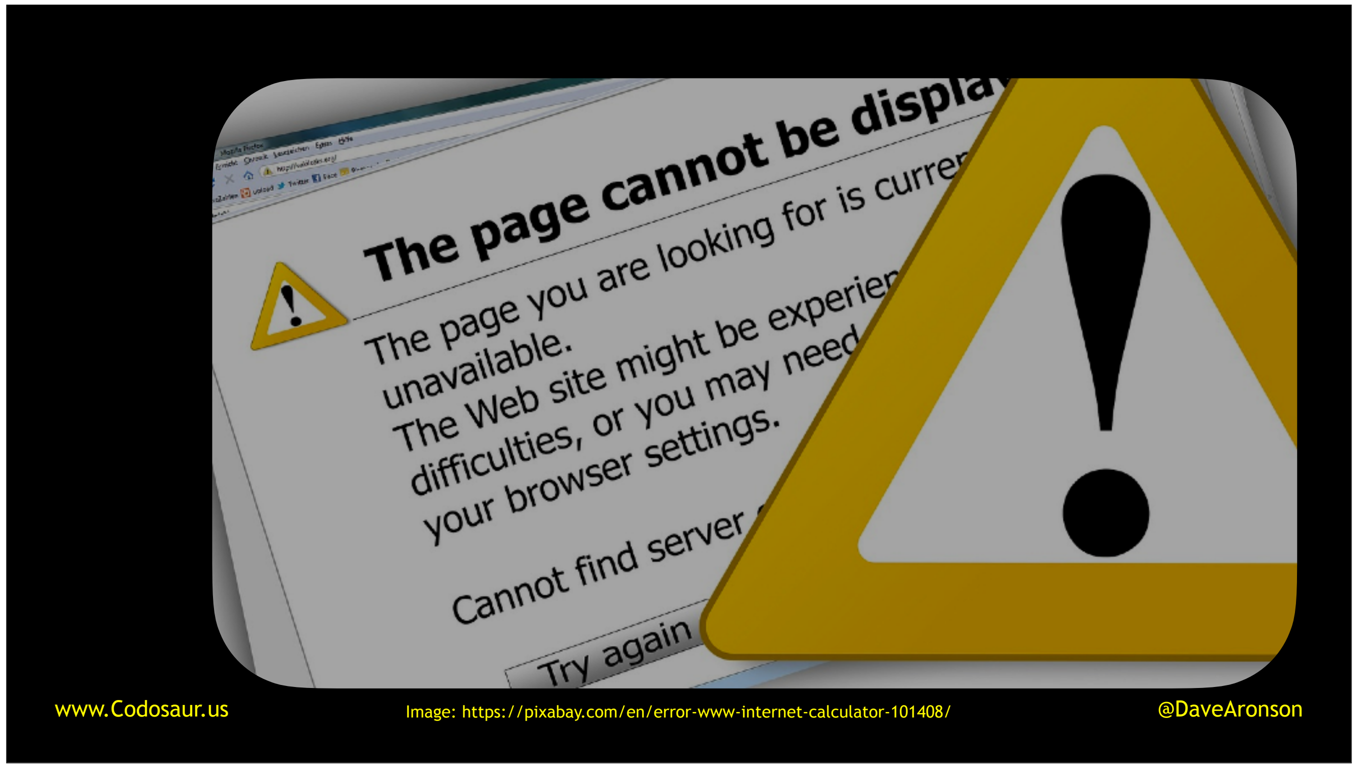

The page cannot be displa
The page you are looking for is curren
unavailable.
The Web site might be experien
difficulties, or you may need
your browser settings.
Cannot find server
Try again

Image: https://pixabay.com/en/error-www-internet-calculator-101408/

. . . or become unavailable when it's not supposed to. Some people would also say it shouldn't become *available* when it's not supposed to, but so long as that doesn't lead to violating one of the other principles, which it does increase the risk of, it's not usually a big deal, maybe a waste of resources.

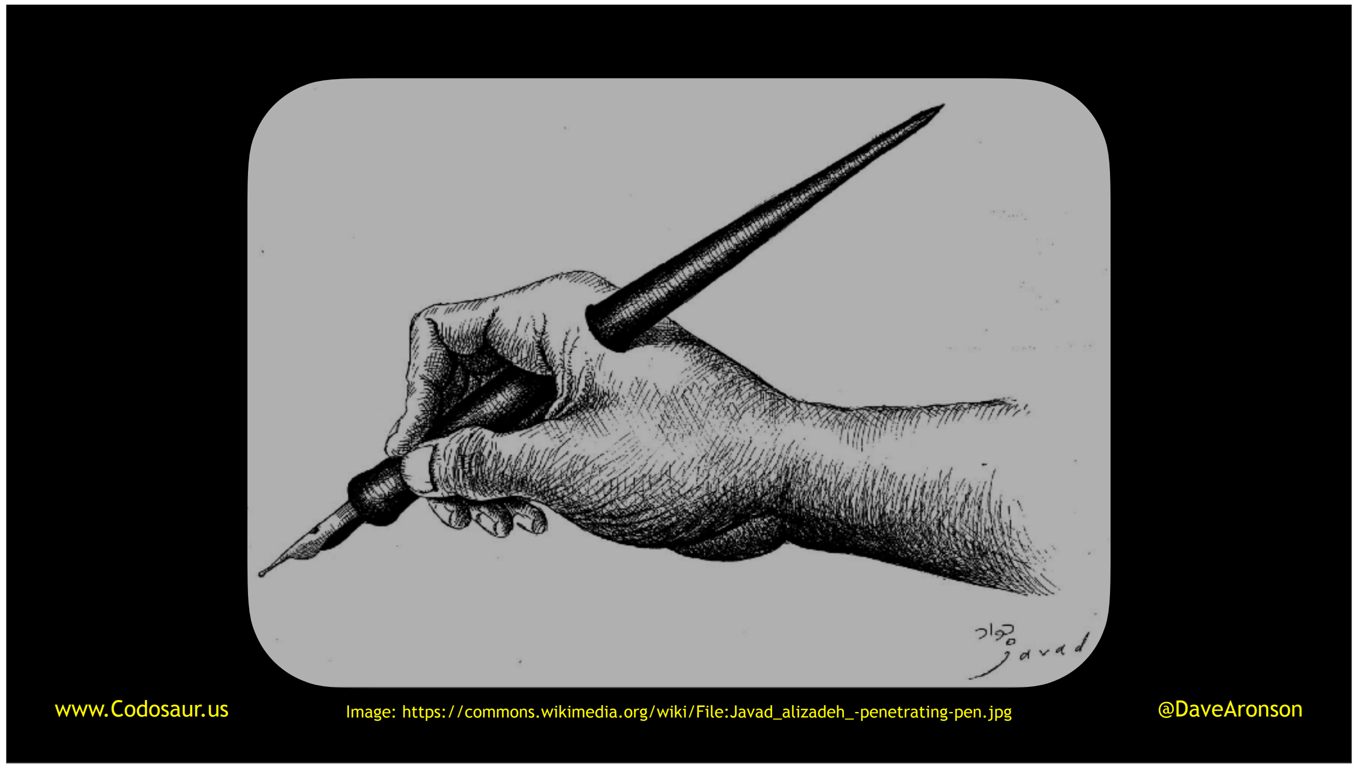
Anyway, a robust system should uphold these principles even when under attack, by someone trying to . . .

www.Codosaur.us
Image: https://pixabay.com/en/star-wars-darth-wader-villain-2463926/
@DaveAronson

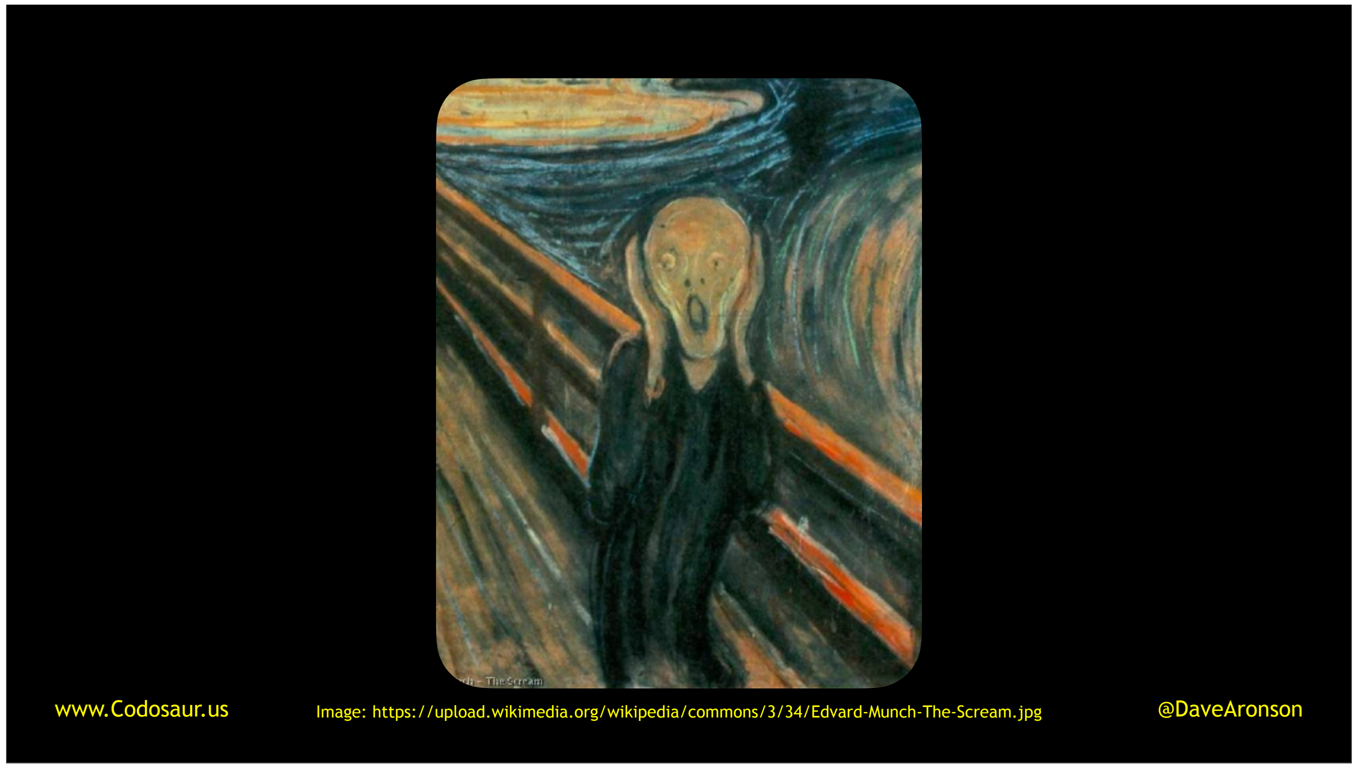
. . . *force* it to violate them.

So how do we achieve all *that?*

Once again, we could bring in the experts, and in this case, that would be . . .


www.Codosaur.us
Image: https://commons.wikimedia.org/wiki/File:Javad_alizadeh_-penetrating-pen.jpg
@DaveAronson

. . . penetration testers, or for short, pen testers. (You can see why I couldn't resist using that image!) The good news is, we don’t have to work for a huge company to use them. Many work for independent computer security companies, that we can hire on a short one-off contract. However, they are usually quite expensive, and disruptive, because they *need* to test the *production* system. (QUICK CUT TO NEXT SLIDE!)



www.Codosaur.us
Image: https://upload.wikimedia.org/wikipedia/commons/3/34/Edvard-Munch-The-Scream.jpg
@DaveAronson

So, once again, we'll usually have to do without them. *But*, we can use some of their tools, especially software such as . . .

www.Codosaur.us

Image: https://creazilla.com/media/clipart/24666/static-on-tv

@DaveAronson

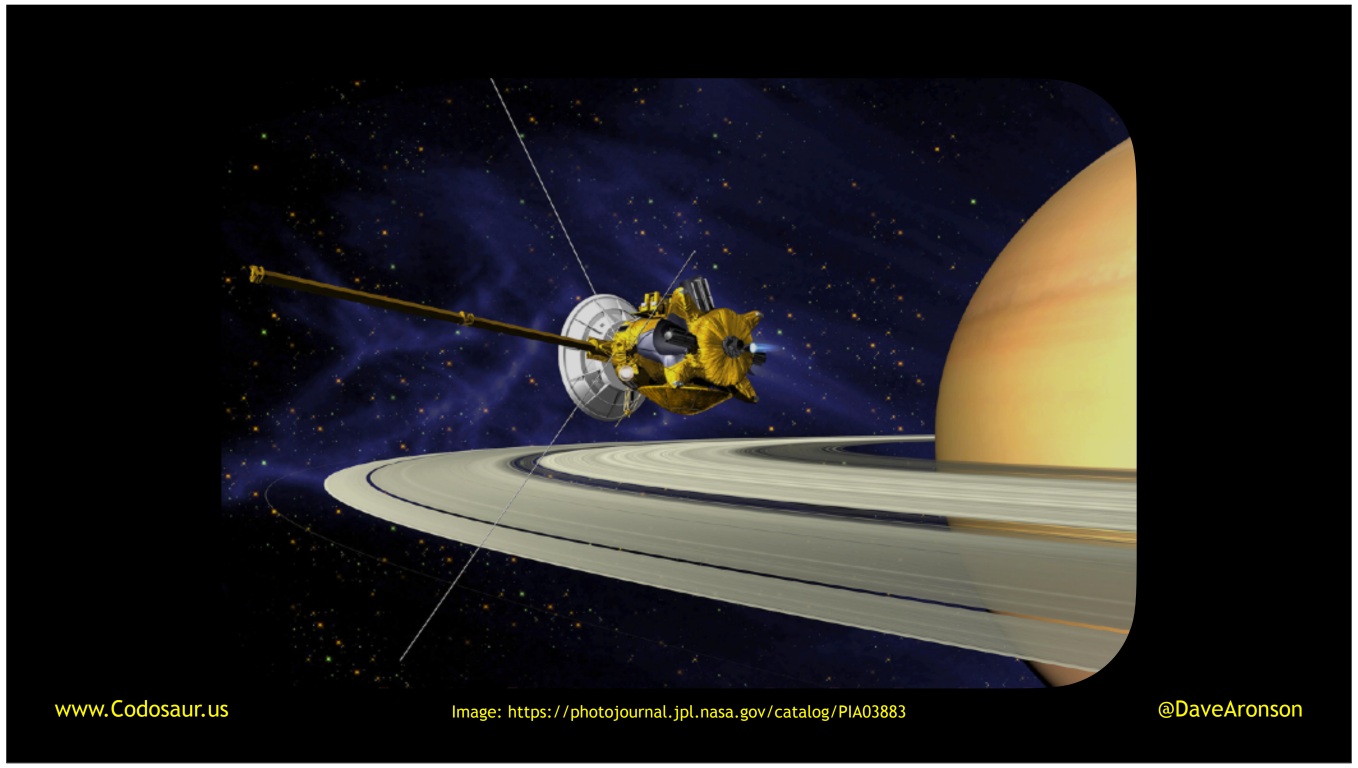
. . . static analyzers, which simulate the execution of our program . . .

www.Codosaur.us

Image: https://www.wannapik.com/vectors/15576 + gradient-filled rounded rectangle made by me

@DaveAronson

. . . fuzzers, which test our program’s reactions to various kinds of invalid inputs, in that "fuzzing" technique I mentioned earlier . . .



. . . and probes, which test our system for vulnerability to specific known attacks.

Many of these are available as open source, or at least free or cheap software.

But even without such software, which our workplaces might be a little bit leery of allowing on their systems and networks, we can still get a long way just by using the *mindset* of a penetration tester, or indeed pretty much any security professional, even in *physical* security. A large part of that mindset is to ask ourselves . . .

. . . what could go wrong. Here the tone of voice is very important, it's not "*What* could go *wrong?*", as though we think nothing could, but almost statement-like, "What could go wrong.", as if to say, "I know a lot *could* go wrong, I'm just trying to list it all out, (LOOK UP) I don't need a demonstration thankyouverymuch!"

For instance, if the system wants the user to . . .

Image: https://pixabay.com/en/folder-icon-file-business-sign-2013220/

. . . type a filename, the user could type it wrong, or type correctly the name of a file they don't have access to, and so on. The program should not crash, or show a mysterious error message like . . .

ENOENT
www.Codosaur.us
Image: https://pixabay.com/en/folder-icon-file-business-sign-2013220/
@DaveAronson

. . . ENOENT, let's have a show of hands, who even knows what that means? Okay, for the rest of you it's basically a very cryptic abbreviation for "file not found", literally E for Error and the rest meaning "no such directory entry", . . .

ENOENT
www.Codosaur.us
Image: https://pixabay.com/en/folder-icon-file-business-sign-2013220/
@DaveAronson

. . . or another cryptic message like . . .

ENOENT
HTTP Code 500
www.Codosaur.us
Image: https://pixabay.com/en/folder-icon-file-business-sign-2013220/
@DaveAronson

. . . HTTP Code 500, let's have another show of hands, who knows what that means? Right, not at all surprising that it's more of you this time, as it's more modern. For the rest of you, it's an *inexcusably* cryptic way of saying "something went wrong on the server" . . .

ENOENT
HTTP Code 500
www.Codosaur.us
Image: https://pixabay.com/en/folder-icon-file-business-sign-2013220/
@DaveAronson

. . . anyway it shouldn't give these cryptic messages just because of a bad filename! And a stack trace is Right Out, not only because it's ugly and generally not useful to the user, but also because it's giving a lot of useful information to any potential *attacker*. So, maybe show one in your development environment, where it can help you, but in production, instead, the system should show a simple, clear, and friendly error message, and let the user try again.

That may sound like a lot, but so far we've only covered innocent mistakes and mishaps. To make it really robust we must make it *secure*, which means we must think like . . .

www.Codosaur.us

Image: https://pixabay.com/en/man-viking-barbarian-machado-1706964/

@DaveAronson

. . . an attacker. We must ask ourselves, what are the system's *weak* points? What can attackers *make happen*, that would get them one step closer to their goal? In what unusual ways can someone get information out of — or *into* — our system? Once we've brainstormed and run out of answers to all these questions, the ones from this slide *and* other things that can go wrong, then for everything we've come up with, we must somehow . . .

www.Codosaur.us Image: https://pxhere.com/en/photo/362236 @DaveAronson

. . . handle it. Yes, that's extremely vague, but how to handle something is going to vary immensely, depending on exactly what it is. If the user typed a bad filename, just let them try again, no big deal, but if we detect that the system is under attack, and sensitive data may be getting corrupted or exfiltrated (a fancy word for copied out), the proper response may well be to shut the whole thing down, and not bring it back up until someone goes into the data center and presses the big green button! In-between, there are many possibilities. Perhaps we can prevent the situation, mitigate the negative effects, or recover from them, perhaps with the help of insurance (though that sort of thing is usually well above our paygrades).

Our software should handle *all* reasonably foreseeable types of problems, from simple user error to system problems like the disk filling up, or external factors, like losing the network connection, as gracefully as practical, while *also* giving as little useful information as possible to potential *attackers*. And then, whatever response we decide on, that requires any technical implementation, we must TEST it, as it is now an important part of . . .

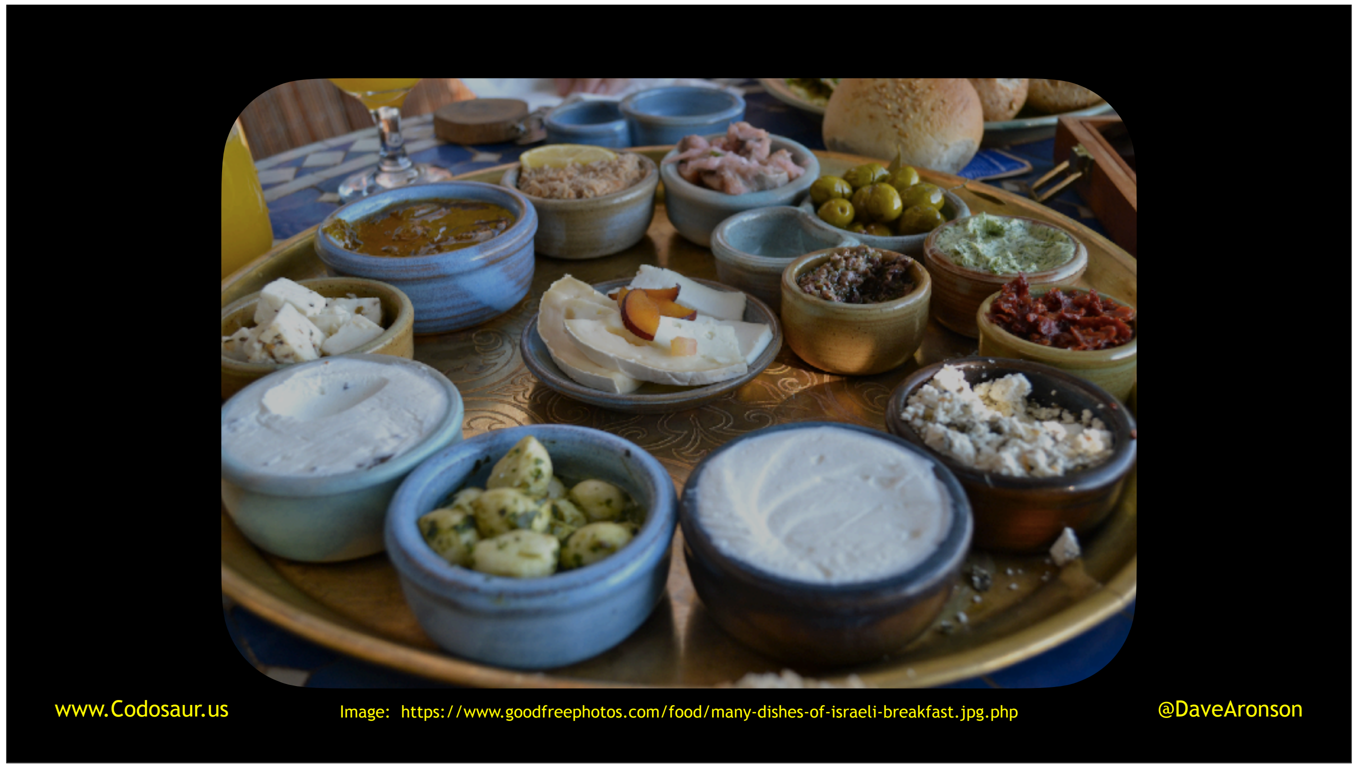
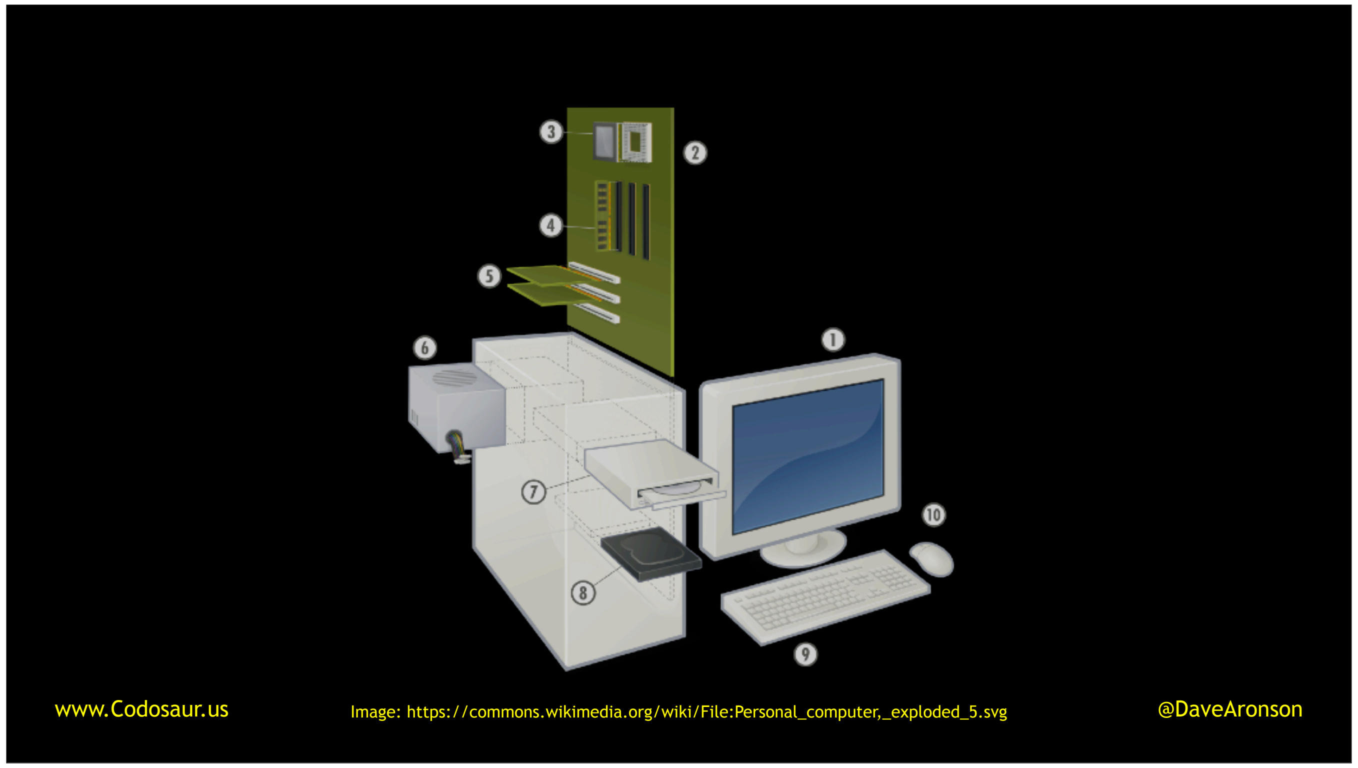

www.Codosaur.us

Image: https://www.goodfreephotos.com/food/many-dishes-of-israeli-breakfast.jpg.php

@DaveAronson

. . . this nutritious breakfast, er, I mean, . . .



www.Codosaur.us Image: https://commons.wikimedia.org/wiki/File:Personal_computer,_exploded_5.svg @DaveAronson

. . . of our system.

Our next aspect is one often seen as a tradeoff with security: . . .

www.Codosaur.us Image: https://www.flickr.com/photos/slgc/24803618672 @DaveAronson

. . . usability, or ease of use. If our software doesn’t have this, our users will become frustrated, and may stop using or recommending our software. That could be *disastrous* for a software vendor, or a software-as-a-service company, especially if there are viable alternatives!

Also, hard-to-use software can lead the user to do the wrong thing. I could give some rather grim examples, like the false alert in Hawai’i in January 2018, about an *incoming nuclear missile,* but on a more humorous note, in mid-April 2024, out of the 150 people collecting endorsements to run for President of Iceland, almost half of them actually just meant to endorse some other candidate already running, not launch their own candidacy. Many of them only found out about the error when someone else endorsed *them!*

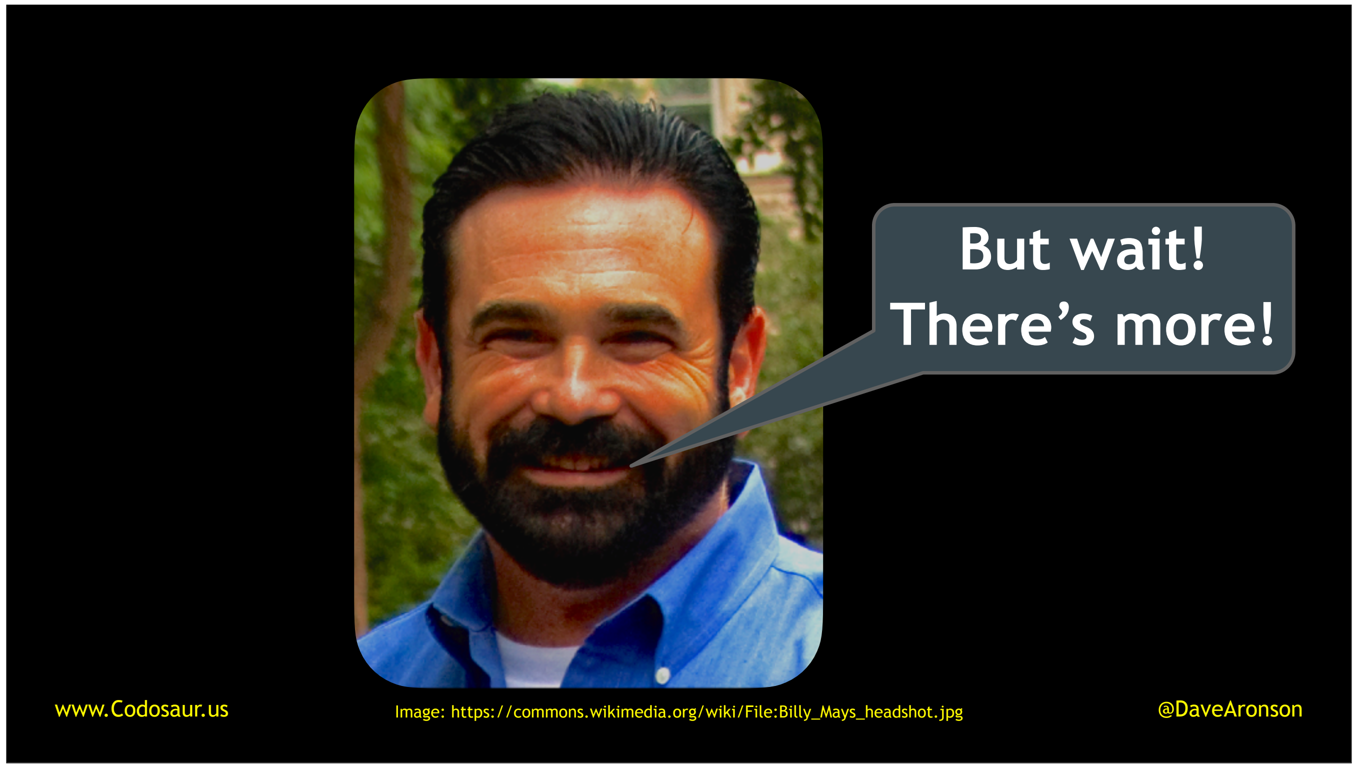
Now once again, the definition says “stakeholders” instead of “users”. So it should *also* be easy for, for example, the *customer service* people to *provide customer service* about the software. It should provide them with easy access to all the data *or documentation* they need, in an easily usable format, whether within the software or somehow exported or logged — but *only with* proper *authorization*, because security. One of the overarching problems is that it’s all interconnected. But anyway, from here on I’m just going to say “user”, which will avoid some confusion because most of the other stakeholders don’t interact *directly* with the software.

Unfortunately, if we Google software usability, we find mostly things about ensuring that users with various challenges can use our software about as well as the rest of us. In other words, accessibility. That’s a good goal in itself, but I’m adding on that it should *easy for everyone* to use, not just *equally difficult!*

To start defining it in more depth: it should be . . .

www.Codosaur.us
Image: https://www.pickpik.com/water-glass-drink-liquid-drinking-glass-glass-material-137270
@DaveAronson

. . . clear, at all times, what the user can do, should do, and *must* do, how they can do it, and . . .



www.Codosaur.us Image: https://commons.wikimedia.org/wiki/File:Billy_Mays_headshot.jpg @DaveAronson

. . . what *else* the software can do, especially any help facilities.

And all of that should be . . .

www.Codosaur.us Image: https://www.flickr.com/photos/jeepersmedia/13943817487 @DaveAronson

. . . *easy to do*, despite any *challenges* the user may be facing. We can *start* with the things that *accessibility* usually addresses, like lack of vision, or color vision (my own main challenge), hearing, or fine motor control, like to use a mouse or trackpad. But there are other whole *types* of challenges we should be aware of. Not everybody is comfortable with computers, knowing even which direction to scroll (which is different on some different operating systems), or when to single or double click, or regular versus right-click, which also may be backwards for people using a left-handed mouse. The user may be illiterate, at least in our character set, like if I tried to use a program that was otherwise fine but the UI is in Chinese — especially including any ways to change it to English. They may lack certain other knowledge, such as cultural references, and especially recent pop culture references. If your UI is divided into sections based on the titles of Shaboozy songs, we 'Boomers (you may have noticed my bifocals) are going to be even more clueless than usual! The users may even be of low *intelligence*. Yes, we may like to joke about stupid users, but statistically, about half of them *will* be below average.

Also again there may be . . .

. . . external factors, like a noisy or shaky environment. Imagine someone using your mobile app or web page, on a small phone, while standing up on a crowded bus in downtown rush-hour traffic! I don't know what that's like here in London, but at least in Washington DC, accurate tapping is not happening.

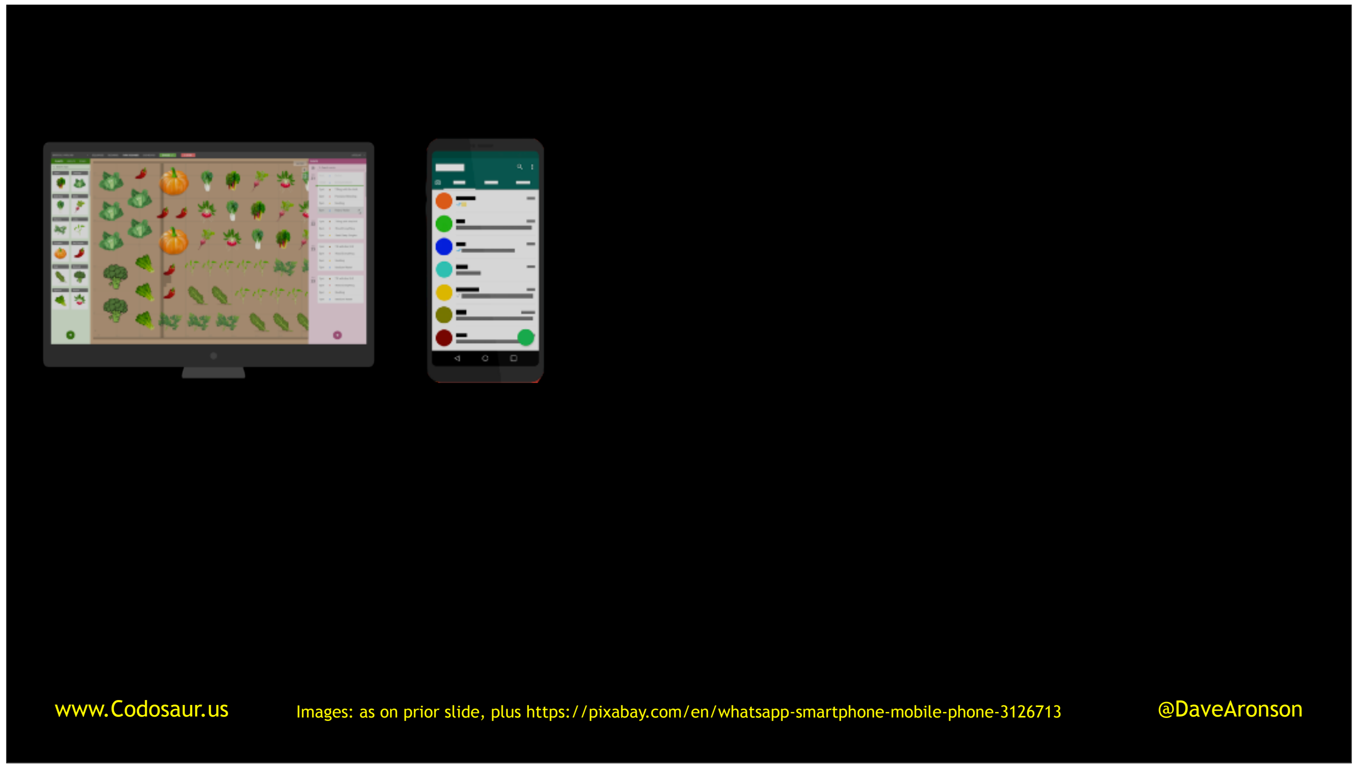
Another often-overlooked part of usability is that ALL software should be usable, whether it’s. . .

www.Codosaur.us

Image: from https://commons.wikimedia.org/wiki/File:FarmBot_Genesis_Web_App_on_Different_Devices.png

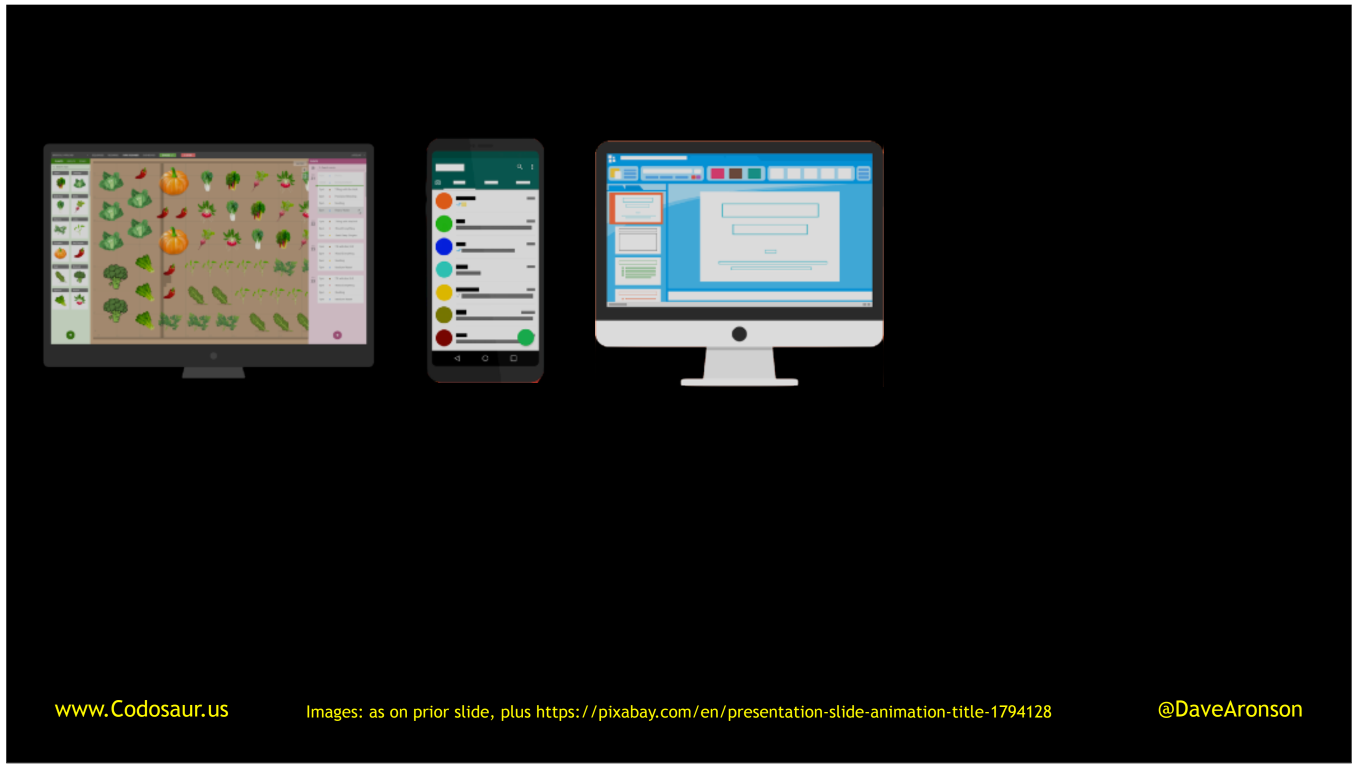
@DaveAronson

. . . a web app . . .



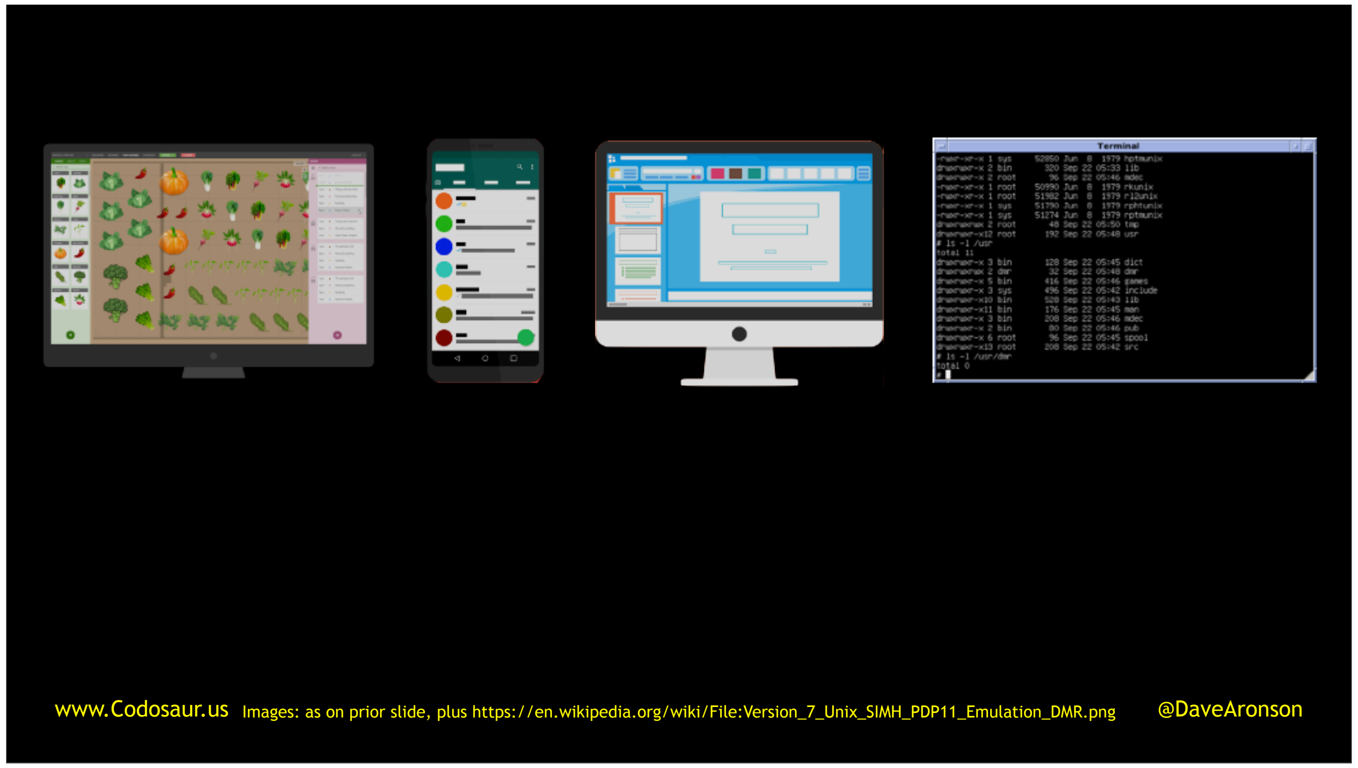
Images: as on prior slide, plus https://pixabay.com/en/whatsapp-smartphone-mobile-phone-3126713

. . . a mobile app . . .



Images: as on prior slide, plus https://pixabay.com/en/presentation-slide-animation-title-1794128

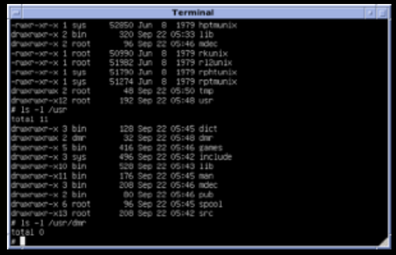
. . . a desktop GUI app, or . . .



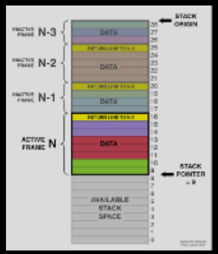
```
-rwxr-xr-x 1 sys     52850 Jun  8  1979 hptmunix
drwxrwxr-x 2 bin       320 Sep 22 05:33 lib
drwxrwxr-x 2 root       96 Sep 22 05:46 mdec
-rwxr-xr-x 1 root    50990 Jun  8  1979 rkunix
-rwxr-xr-x 1 root    51982 Jun  8  1979 rl2unix
-rwxr-xr-x 1 sys     51790 Jun  8  1979 rphtunix
-rwxr-xr-x 1 sys     51274 Jun  8  1979 rptmunix
drwxrwxrwx 2 root       48 Sep 22 05:50 tmp
drwxrwxr-x12 root      192 Sep 22 05:48 usr
# ls -l /usr
total 11
drwxrwxr-x 3 bin       128 Sep 22 05:45 dict
drwxrwxrwx 2 dmr        32 Sep 22 05:48 dmr
drwxrwxr-x 5 bin       416 Sep 22 05:46 games
drwxrwxr-x 3 sys       496 Sep 22 05:42 include
drwxrwxr-x10 bin       528 Sep 22 05:43 lib
drwxrwxr-x11 bin       176 Sep 22 05:45 man
drwxrwxr-x 3 bin       208 Sep 22 05:46 mdec
drwxrwxr-x 2 bin        80 Sep 22 05:46 pub
drwxrwxr-x 6 root       96 Sep 22 05:45 spool
drwxrwxr-x13 root      208 Sep 22 05:42 src
# ls -l /usr/dmr
total 0
# 
```

www.Codosaur.us Images: as on prior slide, plus https://en.wikipedia.org/wiki/File:Version_7_Unix_SIMH_PDP11_Emulation_DMR.png @DaveAronson

. . . a command-line app . . . or . . . an API, be it through . . .



```
-rwxr-xr-x 1 sys     52850 Jun  8  1979 hptmunix
drwxrwxr-x 2 bin       320 Sep 22 05:33 lib
drwxrwxr-x 2 root       96 Sep 22 05:46 mdec
-rwxr-xr-x 1 root    50990 Jun  8  1979 rkunix
-rwxr-xr-x 1 root    51982 Jun  8  1979 rl2unix
-rwxr-xr-x 1 sys     51790 Jun  8  1979 rphtunix
-rwxr-xr-x 1 sys     51274 Jun  8  1979 rptmunix
drwxrwxrwx 2 root       48 Sep 22 05:50 tmp
drwxrwxr-x12 root      192 Sep 22 05:48 usr
# ls -l /usr
total 11
drwxrwxr-x 3 bin       128 Sep 22 05:45 dict
drwxrwxrwx 2 dmr        32 Sep 22 05:48 dmr
drwxrwxr-x 5 bin       416 Sep 22 05:46 games
drwxrwxr-x 3 sys       496 Sep 22 05:42 include
drwxrwxr-x10 bin       528 Sep 22 05:43 lib
drwxrwxr-x11 bin       176 Sep 22 05:45 man
drwxrwxr-x 3 bin       208 Sep 22 05:46 mdec
drwxrwxr-x 2 bin        80 Sep 22 05:46 pub
drwxrwxr-x 6 root       96 Sep 22 05:45 spool
drwxrwxr-x13 root      208 Sep 22 05:42 src
# ls -l /usr/dmr
total 0
# 
```

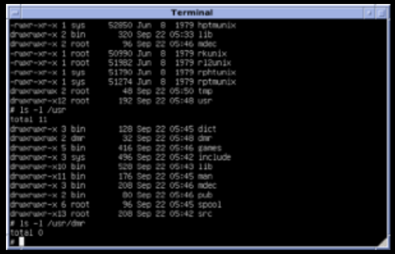


www.Codosaur.us

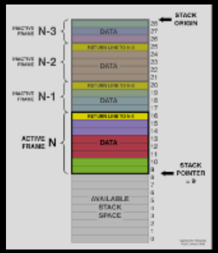
Images: as on prior slide, plus https://commons.wikimedia.org/wiki/File:ProgramCallStack2_en.png

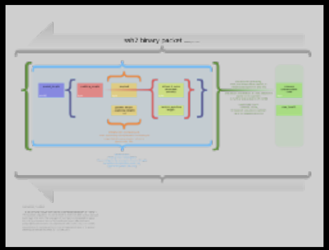
@DaveAronson

. . . function calls like with a library or framework, or a wire protocol, whether . . .



```
-rwxr-xr-x 1 sys     52850 Jun  8  1979 hptmunix
drwxrwxr-x 2 bin       320 Sep 22 05:33 lib
drwxrwxr-x 2 root       96 Sep 22 05:46 mdec
-rwxr-xr-x 1 root    50990 Jun  8  1979 rkunix
-rwxr-xr-x 1 root    51982 Jun  8  1979 rl2unix
-rwxr-xr-x 1 sys     51790 Jun  8  1979 rphtunix
-rwxr-xr-x 1 sys     51274 Jun  8  1979 rptmunix
drwxrwxrwx 2 root       48 Sep 22 05:50 tmp
drwxrwxr-x12 root      192 Sep 22 05:48 usr
# ls -l /usr
total 11
drwxrwxr-x 3 bin       128 Sep 22 05:45 dict
drwxrwxrwx 2 dmr        32 Sep 22 05:48 dmr
drwxrwxr-x 5 bin       416 Sep 22 05:46 games
drwxrwxr-x 3 sys       496 Sep 22 05:42 include
drwxrwxr-x10 bin       528 Sep 22 05:43 lib
drwxrwxr-x11 bin       176 Sep 22 05:45 man
drwxrwxr-x 3 bin       208 Sep 22 05:46 mdec
drwxrwxr-x 2 bin        80 Sep 22 05:46 pub
drwxrwxr-x 6 root       96 Sep 22 05:45 spool
drwxrwxr-x13 root      208 Sep 22 05:42 src
# ls -l /usr/dmr
total 0
# 
```



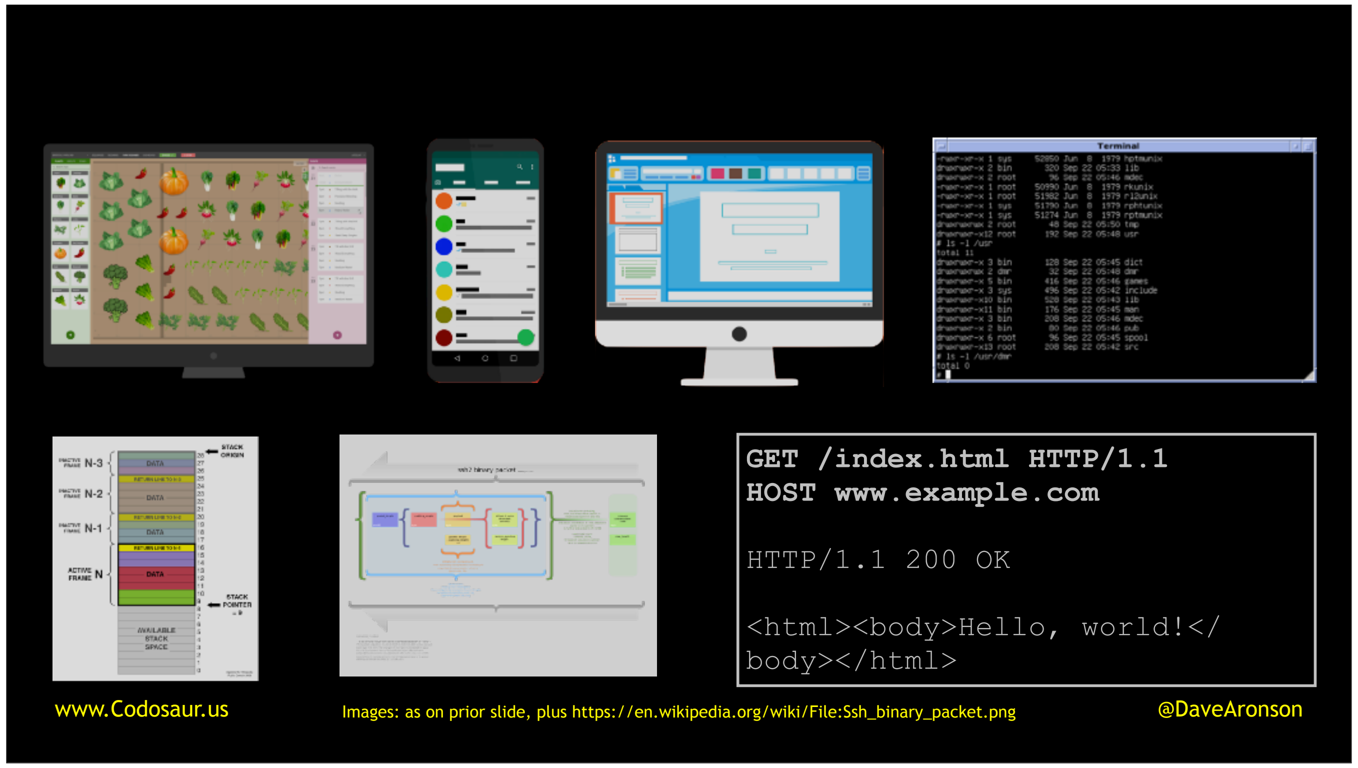


www.Codosaur.us

Images: as on prior slide, plus https://en.wikipedia.org/wiki/File:Ssh_binary_packet.png

@DaveAronson

. . . binary or . . .



```
Terminal
-rwxr-xr-x 1 sys     52850 Jun  8  1979 hptmunix
drwxrwxr-x 2 bin       320 Sep 22 05:33 lib
drwxrwxr-x 2 root       96 Sep 22 05:46 mdec
-rwxr-xr-x 1 root    50990 Jun  8  1979 rkunix
-rwxr-xr-x 1 root    51982 Jun  8  1979 rl2unix
-rwxr-xr-x 1 sys     51790 Jun  8  1979 rphtunix
-rwxr-xr-x 1 sys     51274 Jun  8  1979 rptmunix
drwxrwxrwx 2 root       48 Sep 22 05:50 tmp
drwxrwxr-x12 root      192 Sep 22 05:48 usr
# ls -l /usr
total 11
drwxrwxr-x 3 bin       128 Sep 22 05:45 dict
drwxrwxrwx 2 dmr        32 Sep 22 05:48 dmr
drwxrwxr-x 5 bin       416 Sep 22 05:46 games
drwxrwxr-x 3 sys       496 Sep 22 05:42 include
drwxrwxr-x10 bin       528 Sep 22 05:43 lib
drwxrwxr-x11 bin       176 Sep 22 05:45 man
drwxrwxr-x 3 bin       208 Sep 22 05:46 mdec
drwxrwxr-x 2 bin        80 Sep 22 05:46 pub
drwxrwxr-x 6 root       96 Sep 22 05:45 spool
drwxrwxr-x13 root      208 Sep 22 05:42 src
# ls -l /usr/dmr
total 0
#
```

```
GET /index.html HTTP/1.1
HOST www.example.com

HTTP/1.1 200 OK

<html><body>Hello, world!</
body></html>
```

www.Codosaur.us

Images: as on prior slide, plus https://en.wikipedia.org/wiki/File:Ssh_binary_packet.png

@DaveAronson

. . . textual, or even using an embedded display like a point-of-sale terminal, or a car’s dashboard.

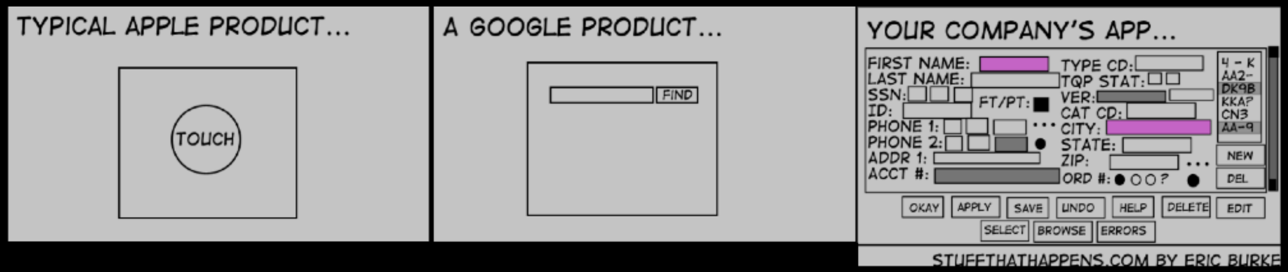
So how do we achieve all this?

Once again, ideally we can bring in . . .

www.Codosaur.us Image: https://commons.wikimedia.org/wiki/File:The_Experts_MET_DT209332.jpg @DaveAronson

. . . the experts. The bad news is, as I said before, the people called “Usability Experts” are mostly really about accessibility. The good news is, we have a wide range of other professions we can get help from! We mainly want a User Experience expert, or maybe a User Interface expert, which is *close* but not *quite* the same thing. But a *designer*, and most of our companies probably have some web or mobile designers on hand, even an old-fashioned *print* graphic designer, also has training in principles of practical visual design that can help us, at least in that aspect of usability.

However, as usual, we’ll often have to do without any help, but we can go a long way by applying some of the *principles* used by these experts. For instance . . .



www.Codosaur.us

Image: http://stuffthathappens.com/blog/simplicity/
(awaiting permission, but Fair Use anyway due to educational purpose)

@DaveAronson

. . . here we see an illustration of the KISS Principle, meaning “Keep It Simple, Stupid”, or if we don’t want to be so negative, “Keep It Super-Simple”. Note the *simplicity* of these stereotypical apps from two highly successful companies with strong reputations for simple ease of use, compared to the cluttered unusable mess from “your company”. I think many of us will recognize some of our own work in that, especially back-enders like myself. Anybody else want to admit that? (PAUSE FOR HANDS)

For the non-visual parts, great progress has been made in API design and documentation standards like REST, GraphQL, Swagger (now called OpenAPI), and so on. This is also helped by test-driven development, TDD, letting us specify the API how we like, while we’re using it to write the *tests*, *before* we’ve implemented it, rather than finding out that it's hard to use *later*, *after* we’ve done all the work of writing the implementation, so it's harder to change. And similarly with BDD, *Behavior* Driven Development, which I consider to be pretty much the same thing, just at a higher, more user-visible level.

Another thing we can do, if the software is something we can use ourselves, is to do so, or . . .

www.Codosaur.us
Image: http://www.usafe.af.mil/News/Article-Display/Article/748008/mwds-cared-for-at-home-station-in-the-field
@DaveAronson

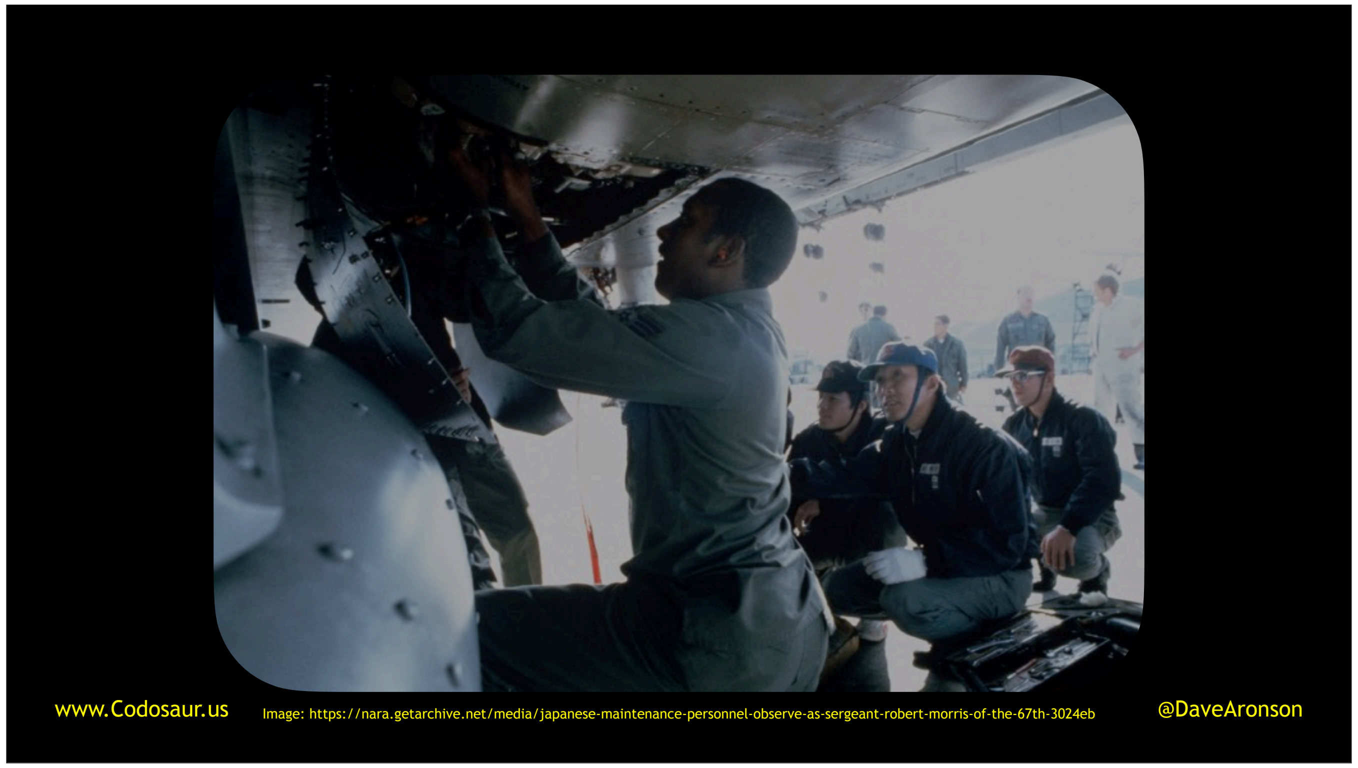
. . ."eat our own dogfood". But remember, if we find our own software easy to use, that does not mean that our users will! We have inside knowledge, that makes it much easier. But if there's anything *we* find difficult or unclear, it will be much worse for our users. So, dogfood it mainly to find the pain points.

Lastly, it may not be as definable and quantifiable as correctness, but a user interface can still be . . .

www.Codosaur.us Image: https://www.flickr.com/photos/eekim/1819059803 @DaveAronson

. . . tested! We can bring in some of our typical users, especially ones that *don't* already know our system, and have them try to do common tasks. We can watch them use it (which is what's going on in this photo), and look for signs, on their faces and screens, of confusion or frustration, or if we're lucky, satisfaction, or maybe even happiness. Afterward, *ask them* what they found hard or easy, unclear or obvious? Then fix their pain points, and do more of the good parts.

The next aspect is the one we developers probably usually think about most, well maybe second behind Correctness: . . .


www.Codosaur.us
Image: https://nara.getarchive.net/media/japanese-maintenance-personnel-observe-as-sergeant-robert-morris-of-the-67th-3024eb
@DaveAronson

. . . maintainability. If our software doesn't have this, then changes take longer, and are more likely to introduce bugs, and developer headaches. Delays could make the company miss opportunities. Bugs damage the company's reputation. Developer headaches are bad enough for us, but for the company, they could make key personnel quit in frustration. Anybody else been there, as either the quitter or a survivor who had to pick up the slack?

We'd probably all agree that the basic concept is that "maintainable" software is easy to change. (Thank you, Captain Obvious! I think that's him, in the back there.) But I'm going to add, that it's easy to change, *with* . . .

www.Codosaur.us

Image: https://pxhere.com/en/photo/615255

@DaveAronson

. . . low *chance* of error (we don't want a *dicey* situation), and . . .

www.Codosaur.us
Image: https://pixabay.com/en/potatoes-fear-horror-pot-cook-3119211/
@DaveAronson

. . . low *fear* of error, even for . . .

www.Codosaur.us

Image: https://pxhere.com/en/photo/796734

@DaveAronson

. . . a novice programmer, who is also . . .

NEW GUY
www.Codosaur.us
Image: https://commons.wikimedia.org/wiki/File:New_Guy_(5895483627).jpg
@DaveAronson

. . . new to our project.

Now how do we achieve all this? For better or worse, the vast majority of software engineering advice is aimed squarely at this. So, rather than expound on lots of generic principles like High Cohesion, Low Coupling, good naming, Single Responsibility, and so on, I'm going to stick to my theme and tell you how . . .

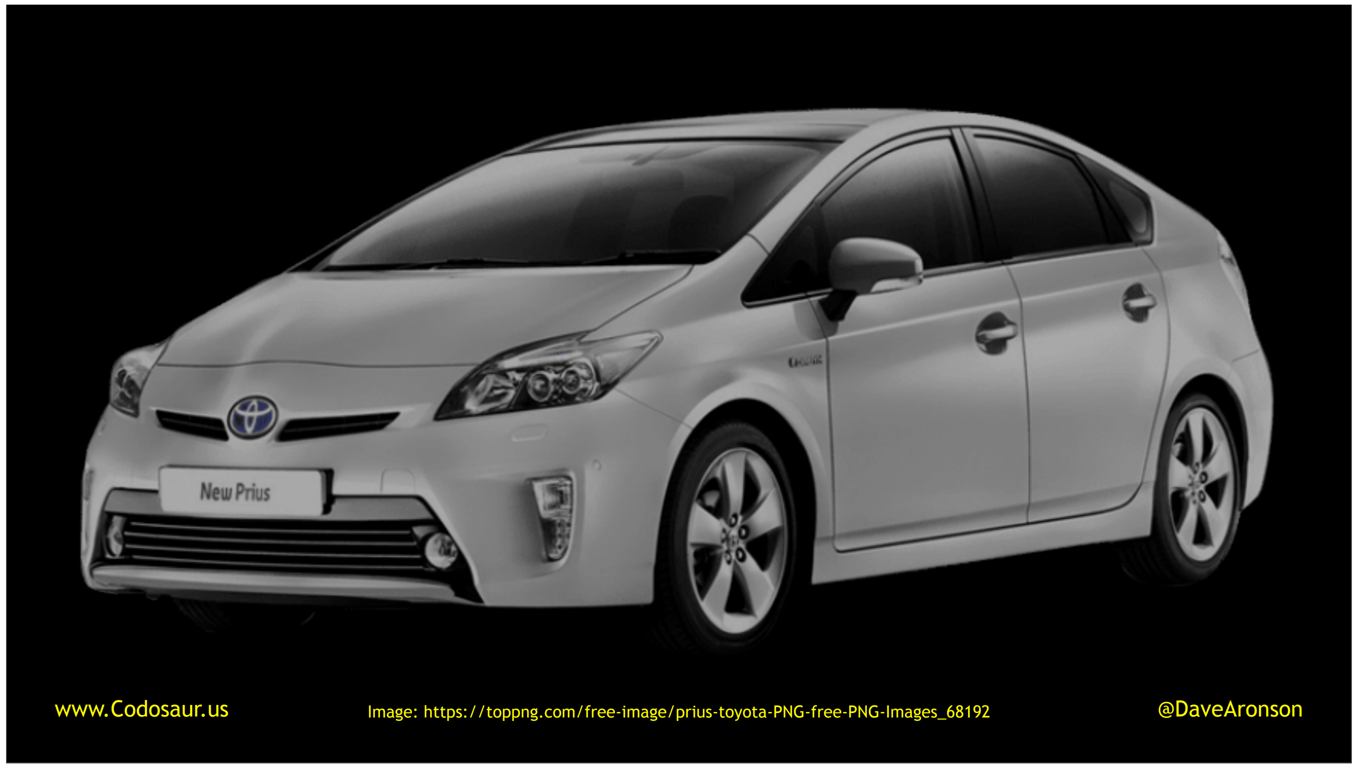
. . . testing can help with maintainability. Some of you may already be in the habit of using tests as a sort of *documentation* of how the code should be *used*. This is *especially* useful in languages such as Elixir or Python, that support doctests, which are tests built right into the documentation. But also, all of the tests we wrote to verify any *previous* changes we made, whether adding a feature or fixing a bug or whatever, form a *regression* test suite, to catch anything we break, that used to work. Just *knowing* that that is *there*, as a *safety net*, will reduce our *fear* of error. And *that* will allow us to progress at a quick and steady pace, with a clear and focused mind, “I think I can, I think I can” … rather than creeping along … slowly … and erratically … because we’re terrified … of breaking something accidentally … and not discovering it … until the users start to complain. And *that* speedup is why I mentioned fear at all.

There are also lots of . . .

www.Codosaur.us
Image: https://pixabay.com/photos/tools-work-repair-hammer-2145770/
@DaveAronson

. . . tools we can use, like linters, complexity analyzers, and just cranking up the warnings on our compilers or interpreters. These will give us lots of hints how to improve our code, mainly in its maintainability, but also occasionally uncovering subtle bugs. That used to be one of my favorite ways to find opportunities to improve a codebase I inherited, often with warnings turned way down low, especially in C because it's so easy to make C generate way too many warnings. Anybody else ever try that trick, and find some hidden nastiness?

For the final aspect, software should . . .


New Prius
www.Codosaur.us
Image: https://toppng.com/free-image/prius-toyota-PNG-free-PNG-Images_68192
@DaveAronson

. . . go easy on resources. If we don't have this, then our programs may run slowly, or make the users buy more resources. They could clog the network and make *everybody's* machine *seem* slow, or even crash machines by running out of memory or disk space, or drain other resources. Mainly we know about techical resources, like CPU, RAM, bandwidth, and my own pet peeve lately, screen space, but there are other kinds, such as the user's *patience* and *brainpower*, and the company's money. I mean both the company that may be buying or licensing our software, *and* the one *paying us* to write it in the first place!

So how do we achieve efficiency? There are many kinds of resources, and many ways each can be abused, so there are many many different kinds of inefficiency, but for the sake of this presentation I'm just going to focus on fixing the most obvious and common kind: slowness.

It's quite common to have a program run slowly, then we stare at the code, spot where we think it's inefficient, spend a long time optimizing that little piece, run the program again, and . . . it's still slow! Right? Has anybody else been there? (PAUSE!) Don't do that!

www.Codosaur.us

Image: https://pixabay.com/en/diet-calorie-counter-weight-loss-695723/

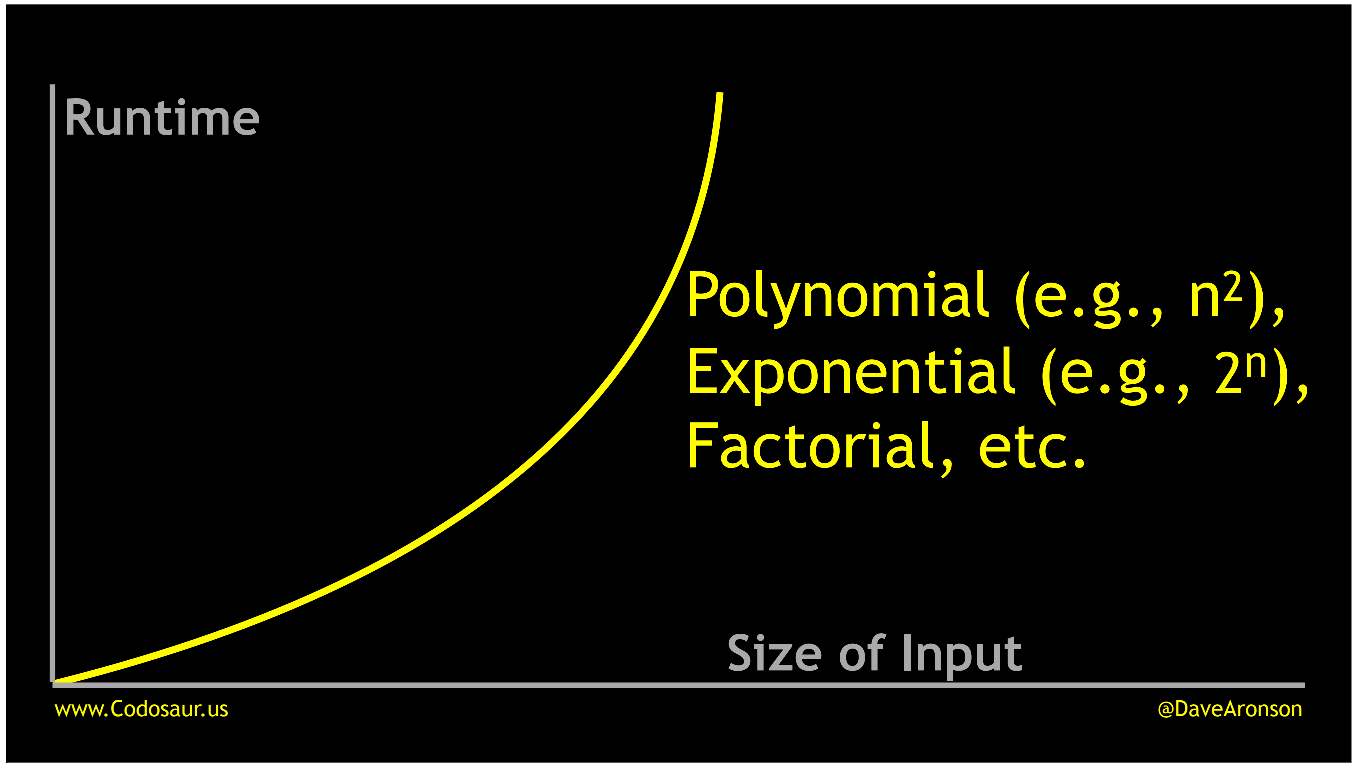
@DaveAronson

Measure it instead! It turns out, that humans aren't really very good at spotting the inefficiencies (surprise surprise!), especially when the inefficiencies are buried deep down in a dependency tree! But there are *profilers* and *packet capture programs* and so on, that will tell us *exactly* where, or at least when, we're using too much CPU, RAM, bandwidth, etc.

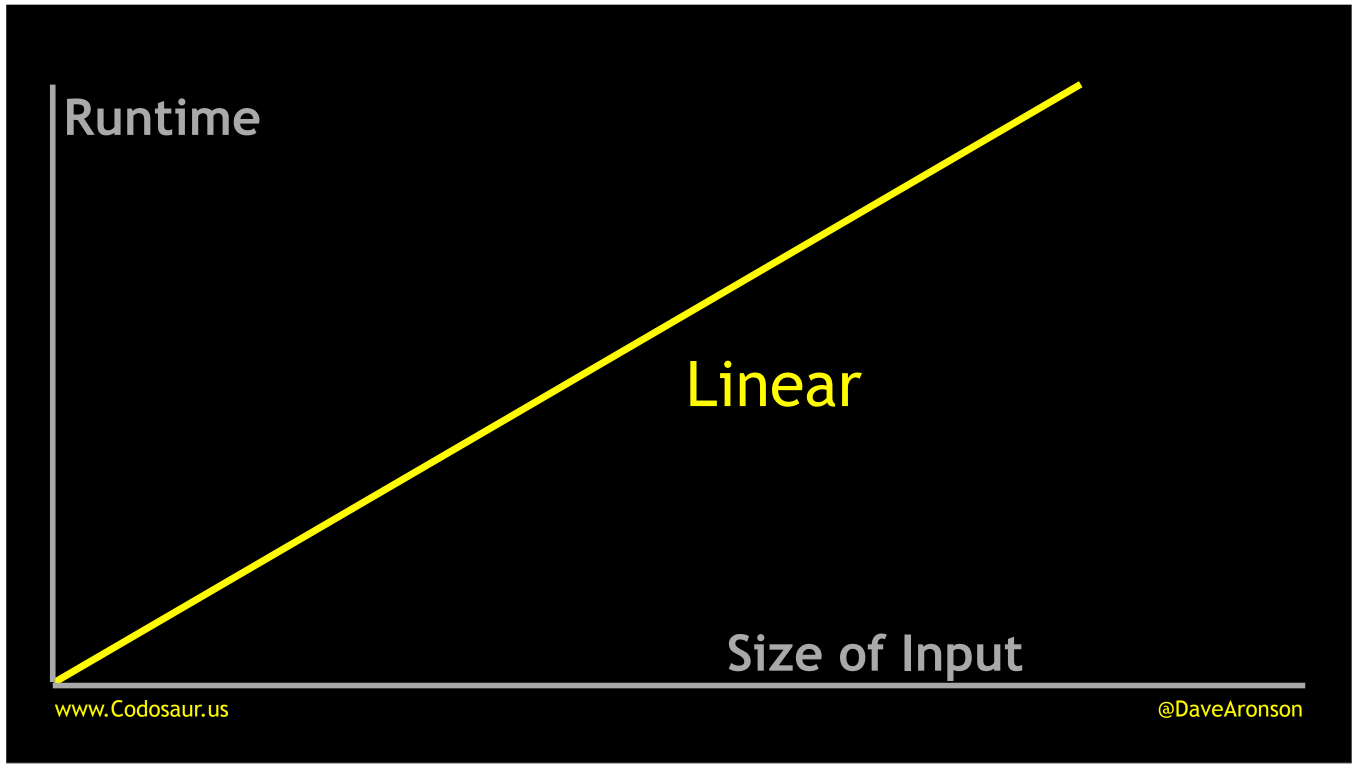
Once we've found *where* or *when* it's slow, though, there's still the question . . .

Image: https://pixabay.com/en/snail-animal-fun-surprised-160313/

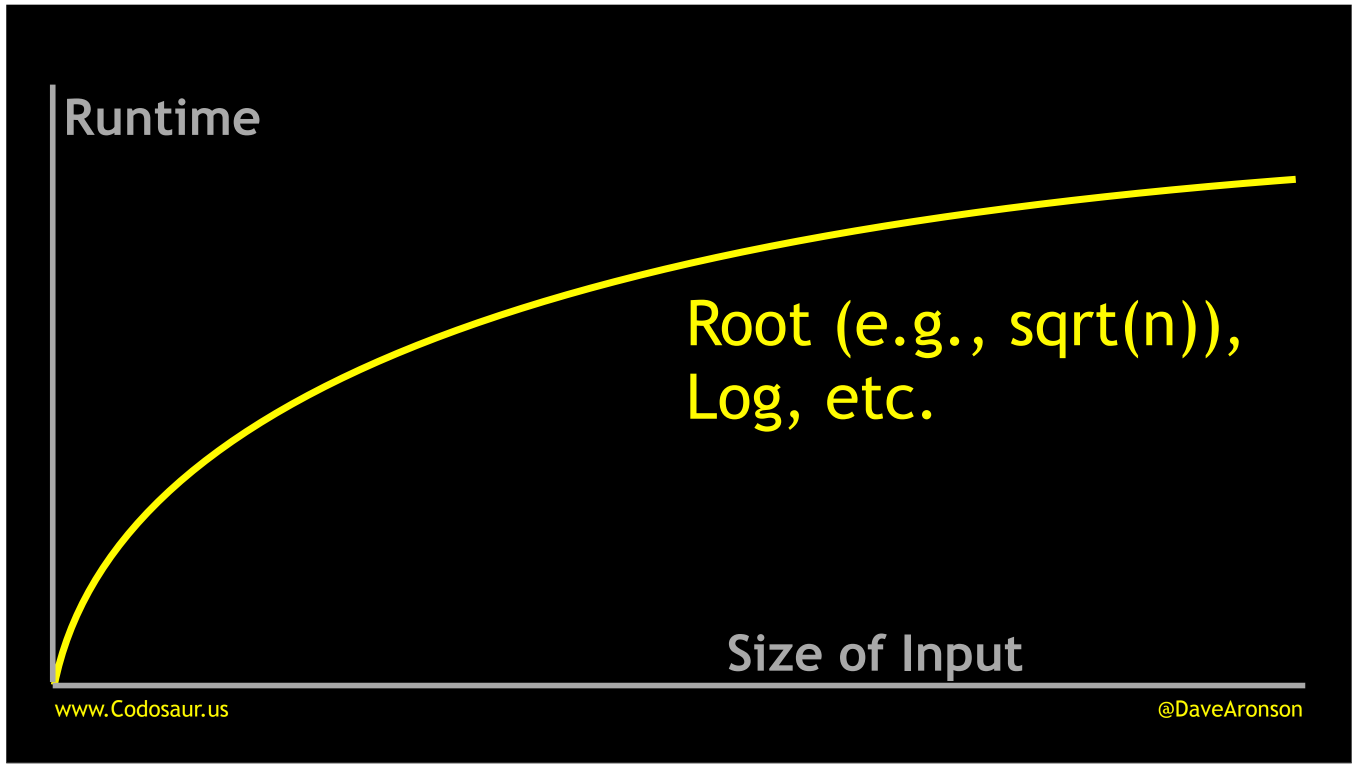
. . . *why* is it slow? Certain kinds of programs tend to have certain kinds of problems. For instance, a distributed system may be doing too much communication, or doing a reasonable amount of it but doing it over a slow network. A database-driven system may have an inefficient query, either inherently inefficient, like the infamous N+1 query problem, or because it's being run against an inefficient underlying data model. Maybe it's missing some useful indices when reading, or *updating too many* indices when *writing*. But in the much broader general case, usually the problem is either something *architectural*, which is more complex than I want to get into right now, or a *bad algorithm*. Maybe we're using something with a . . .



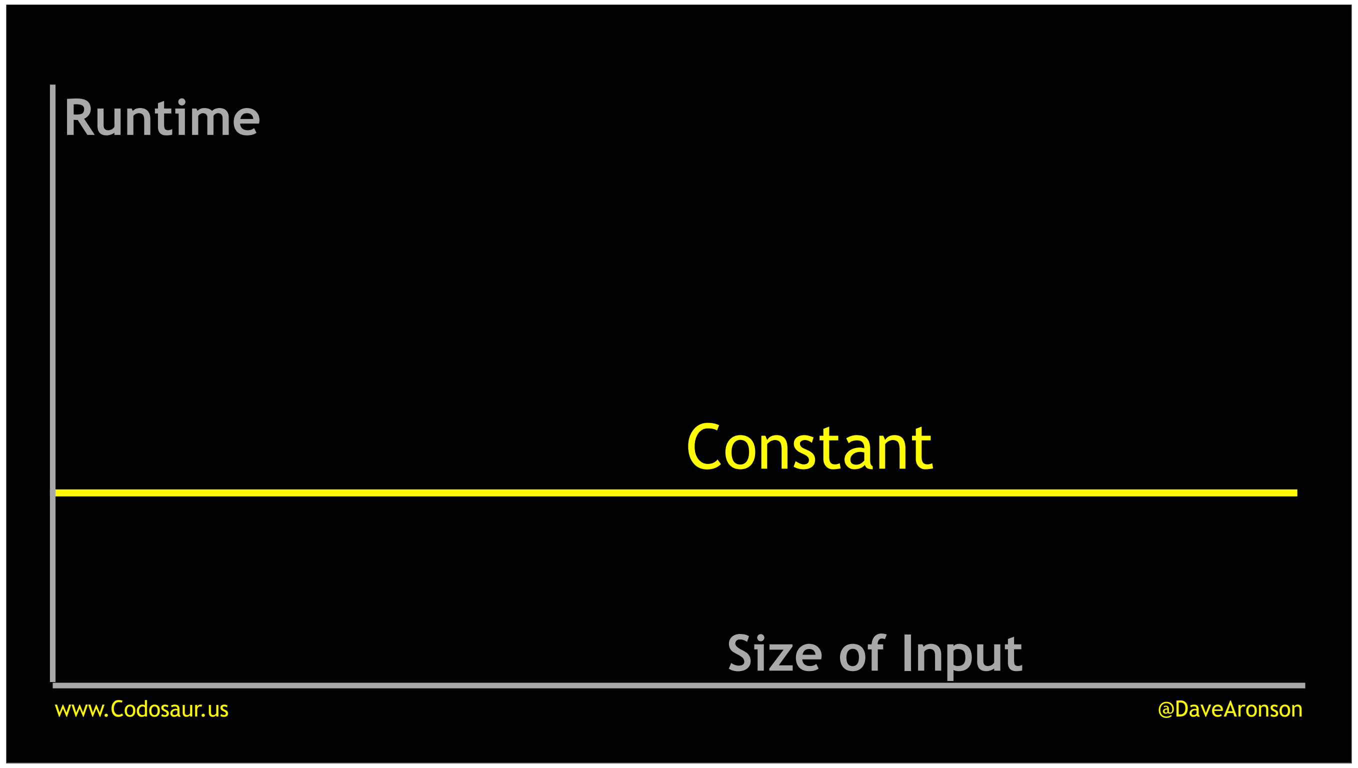
. . . really bad runtime, like polynomial, exponential, maybe even factorial, when looking at the problem from a slightly different angle, could let us use a better algorithm, such as one with . . .



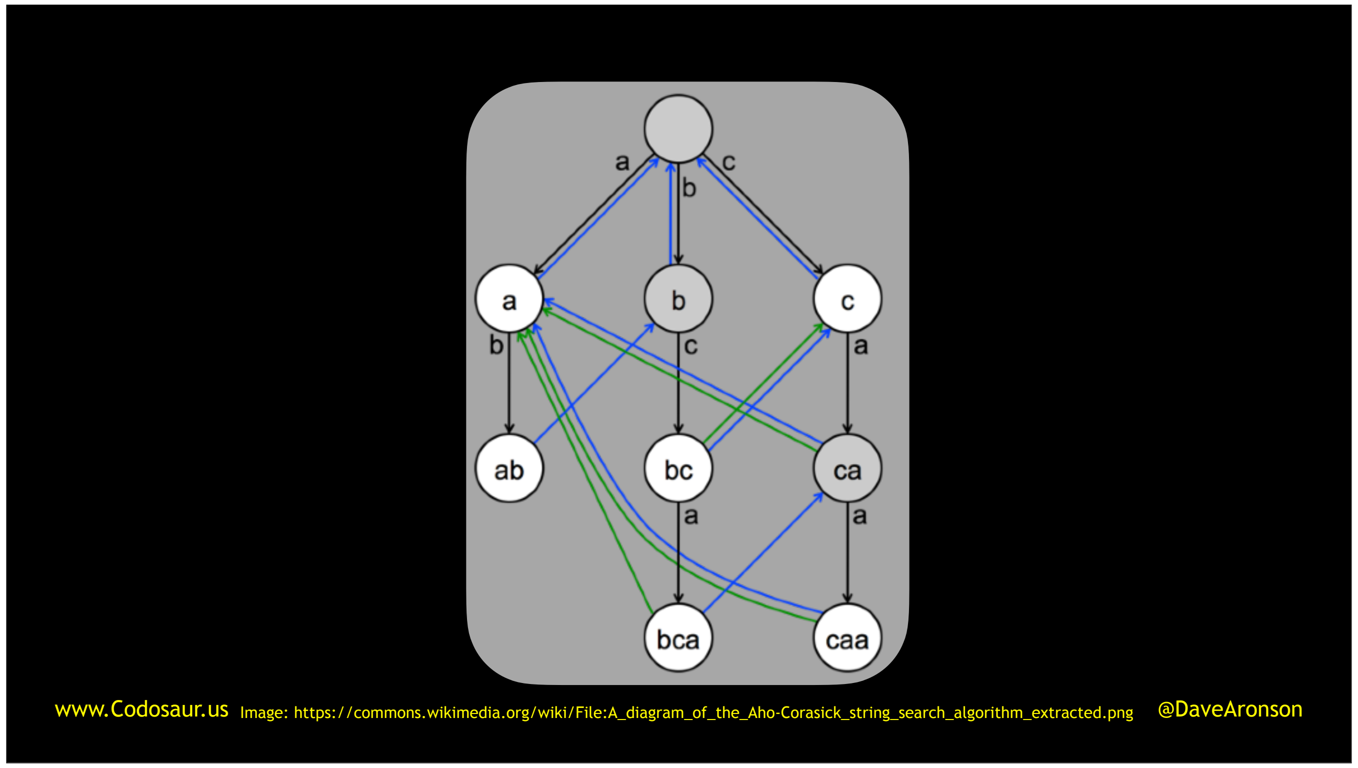
. . . linear runtime, or better yet . . .



. . . *root* or *logarithmic* runtime, or maybe even the holy grail of . . .



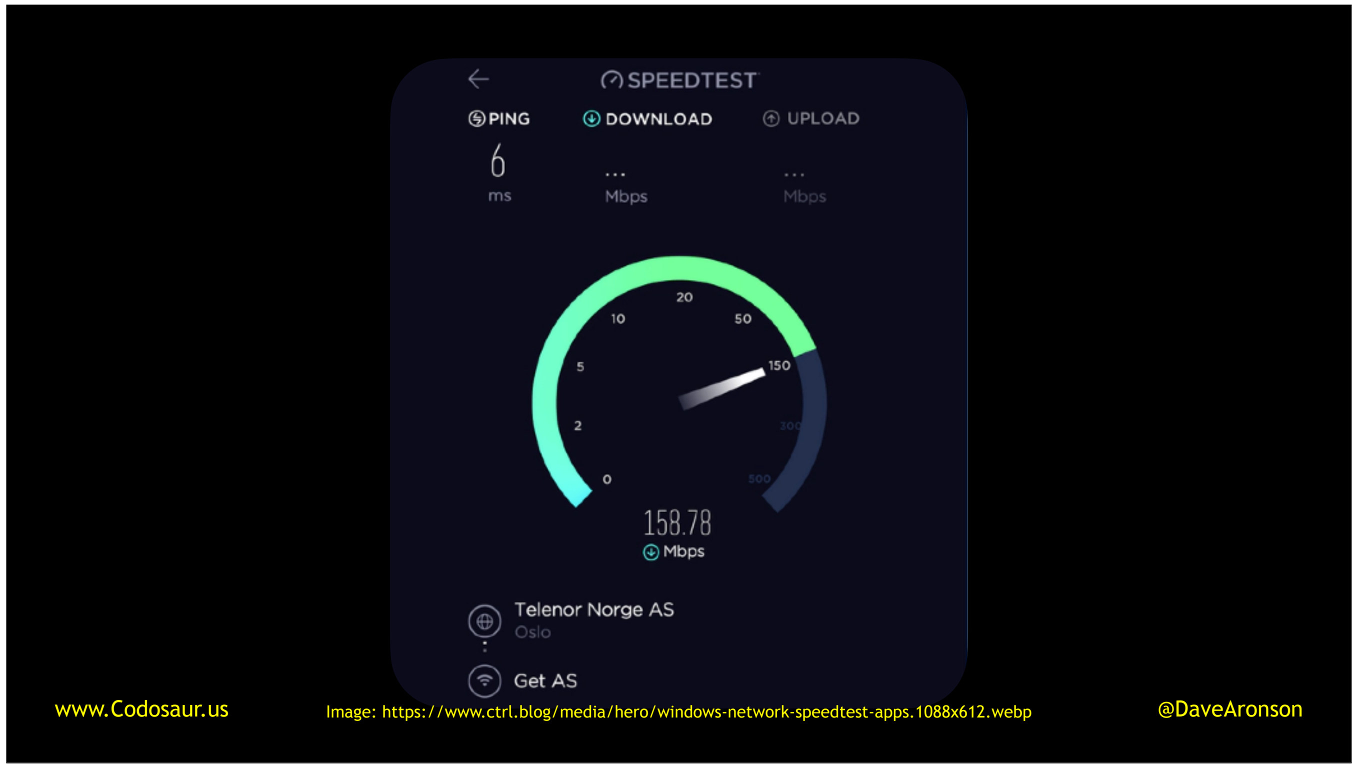
. . . constant time. Perhaps we’re using . . .


a
b
c
a
b
c
b
c
a
ab
bc
ca
a
a
bca
caa
www.Codosaur.us
Image: https://commons.wikimedia.org/wiki/File:A_diagram_of_the_Aho-Corasick_string_search_algorithm_extracted.png
@DaveAronson

. . . a bad *data structure,* and *that* is *forcing* us to use a bad algorithm.

The upshot is that, no matter how sick we may be of hearing about data structures and algorithms this, and DSA that, and their abuse in hiring practices based on Leetcode and other such trick question gotchas, we do *need* to be familiar with at least the common basic ones. That way we can *recognize* them when we see them in real-world problems, *analyze* and *compare* their demands on our resources, and *choose* and *change* and *combine* them. That way, we can use solutions that have stood the test of time, sometimes with ready-made implementations that might well tested and maybe even optimized, as opposed to building our own square wheels.

Once it’s fast *enough*, we can slap a . . .



www.Codosaur.us Image: https://www.ctrl.blog/media/hero/windows-network-speedtest-apps.1088x612.webp @DaveAronson

. . . *performance test* around it (you knew I had to advocate doing some kind of testing eventually), to prevent that kind of regression.

Now that we've explored ACRUMEN fairly thoroughly, I'll answer one last frequently asked question: didn't I . . .

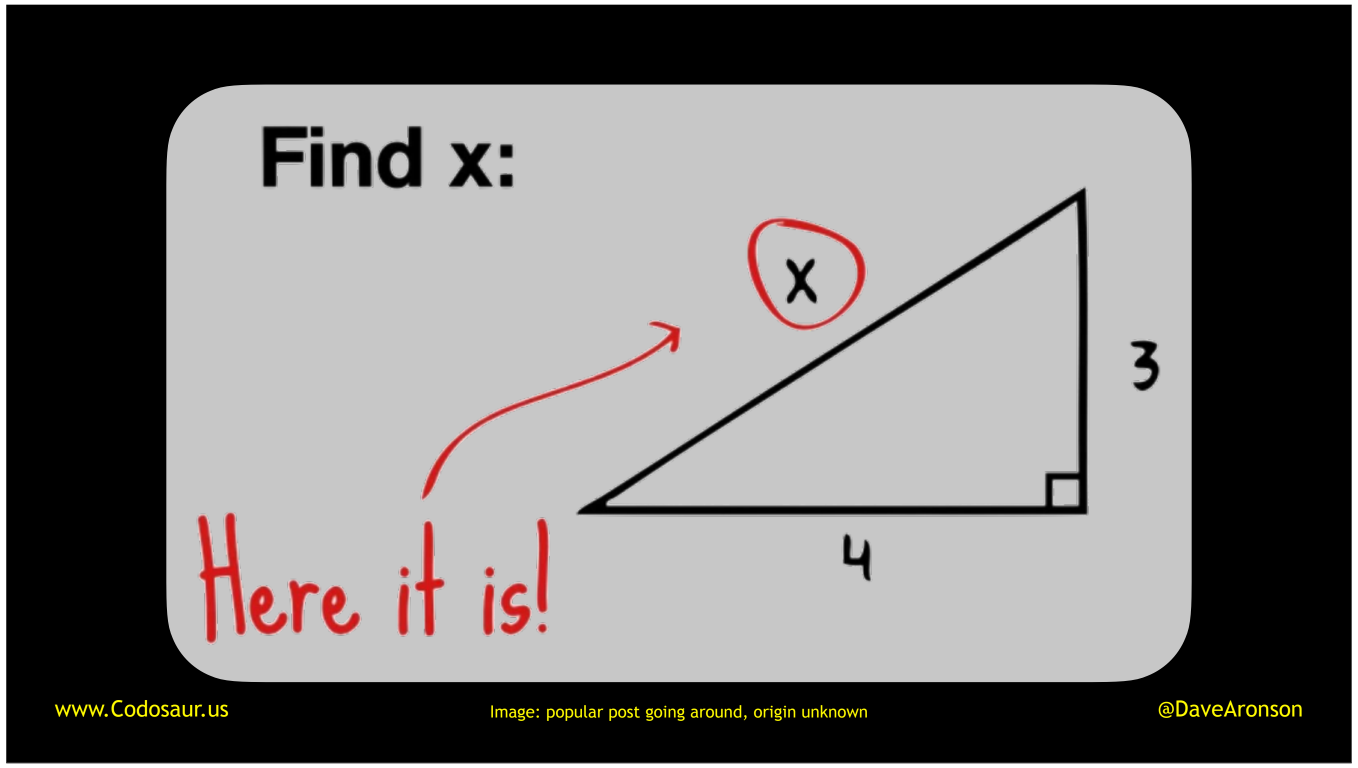


Image: popular post going around, origin unknown

. . . find *any* definitions worth using? Well . . . sort of. Not at first, while I was still *formulating* ACRUMEN. There were some that had useful ideas at a top level, but then went into way too much detail underneath. For instance, there was one that I think was from . . .

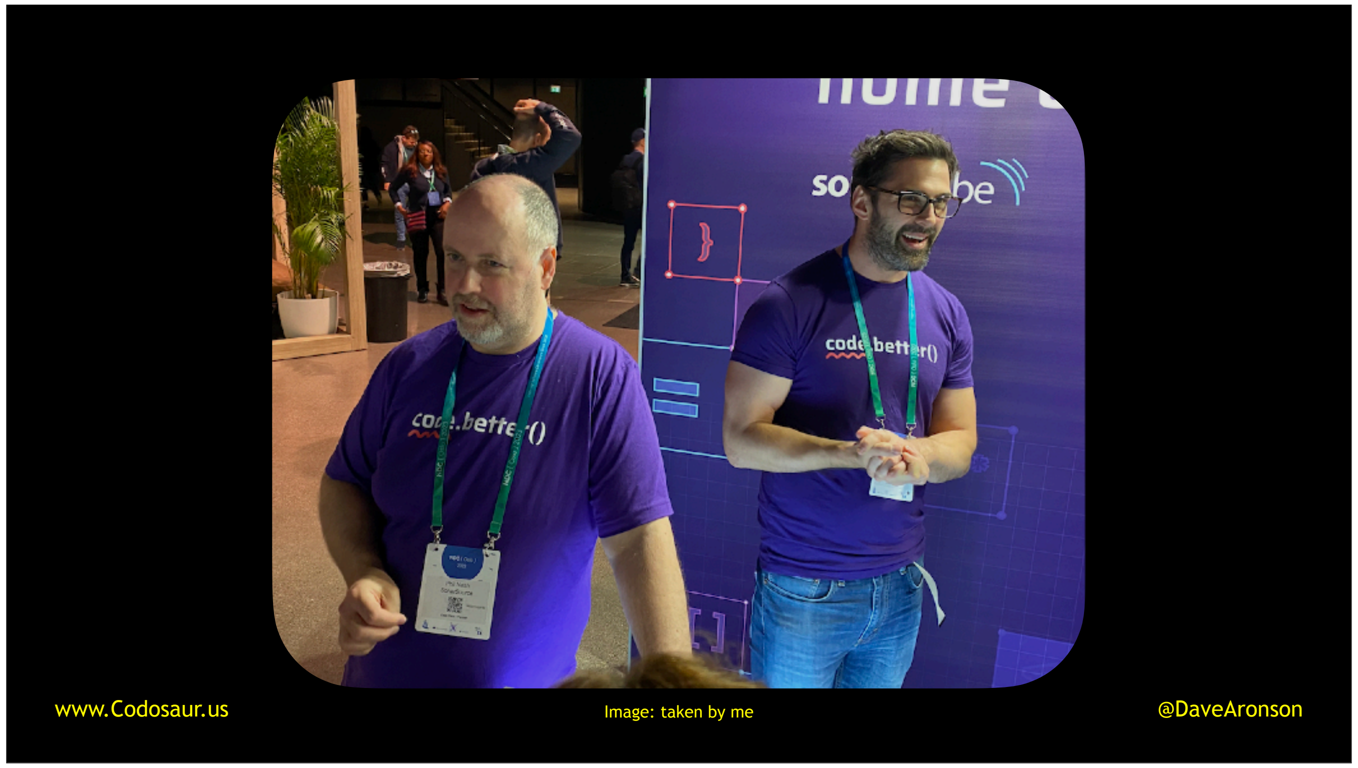
www.Codosaur.us

Image: https://www.thedetroitbureau.com/wp-content/uploads/2020/11/Ford-Quality-is-Job-1.jpg
(not licensed, but used under Fair Use educational provisions)

@DaveAronson

. . . Ford Motor Company, in the form of a list of only *five* aspects . . . but then they subdivided it into *sub*aspects, trying to list every way that software could be good or bad, and pigeonholing each one into one of their five main aspects, when in reality it seems to me that many of them span multiple top-level aspects. Between how that just didn't really work, and how that made it longer and more complex, that knocked it out of the running as a good *realistically applicable* definition. There were several other systems that did that, and similar problems, though in fact it was from such systems that I gained an appreciation for the importance of usability. As I mentioned before, I'm much more of a back-end developer, with more emphasis on correctness, robustness, efficiency, and maintainability. (And to complete the set, like most of you, Appropriateness was often "above my paygrade" to worry about, except maybe an occasional bit of pushback when I really didn't see it.)

However, *after* I had already been writing and speaking about ACRUMEN, *then* I *finally* found some definitions I liked — though not enough to make me stop. Mainly, those of you who go to a lot of conferences might know . . .

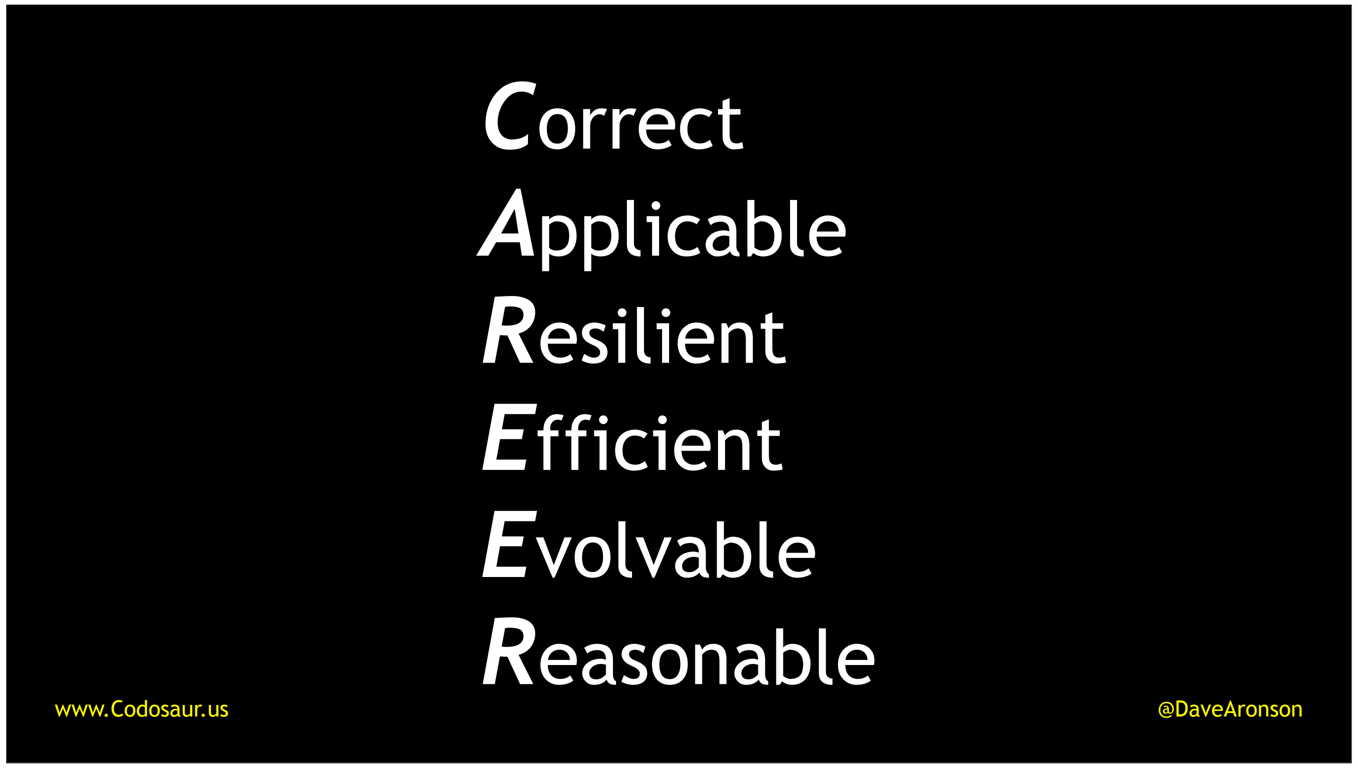


www.Codosaur.us

Image: taken by me

@DaveAronson

. . . Phil Nash. These guys are *both* frequent international software development conference speakers named Phil Nash, and yes, hilarity often ensues, especially when they were working for the same company for a while. Especially since they're both from the UK, though the one on the right now lives in Australia. The on the left was speaking at a conference a few years ago, just over in Bristol, where I was also speaking. (Not doing this talk, but another one.) A few weeks before the conference, I was looking through the agenda, and saw that he was going to speak on *his* definition of software quality… which turned out to be a list of aspects… six of them… summed up in an acronym… no not ACRUMEN but . . .



Correct

Applicable

Resilient

Efficient

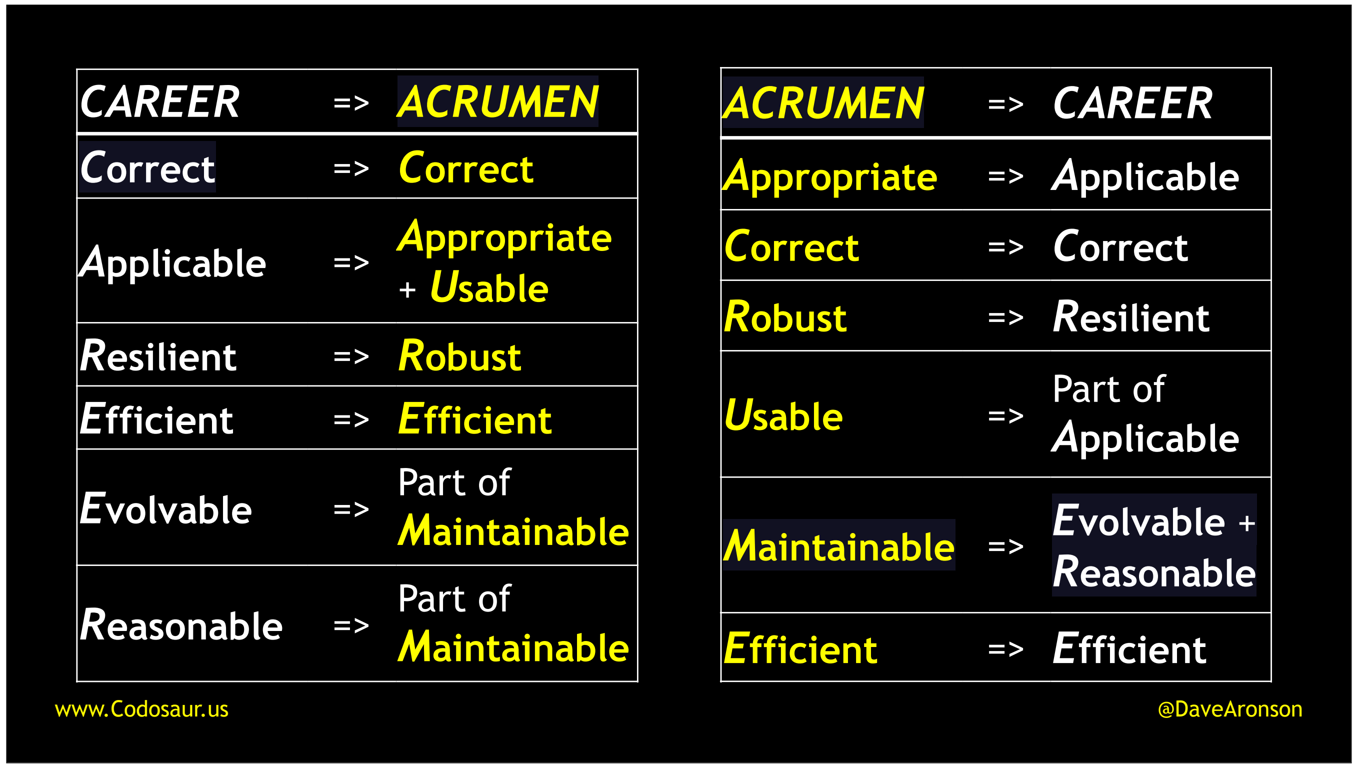
Evolvable

Reasonable

www.Codosaur.us

@DaveAronson

. . . CAREER, so there's no plagiarism going on there. After watching some of his videos on it, it was clear that it said mostly the same thing as ACRUMEN, which is how I knew it was good. (PAUSE FOR LAUGHS) No, seriously, it was a bit of a relief to see someone who was already well-respected on the conference circuit, thinking along similar lines. It may have been a stupid idea, but at least I wasn't the only one to have it! The main difference is that . . .



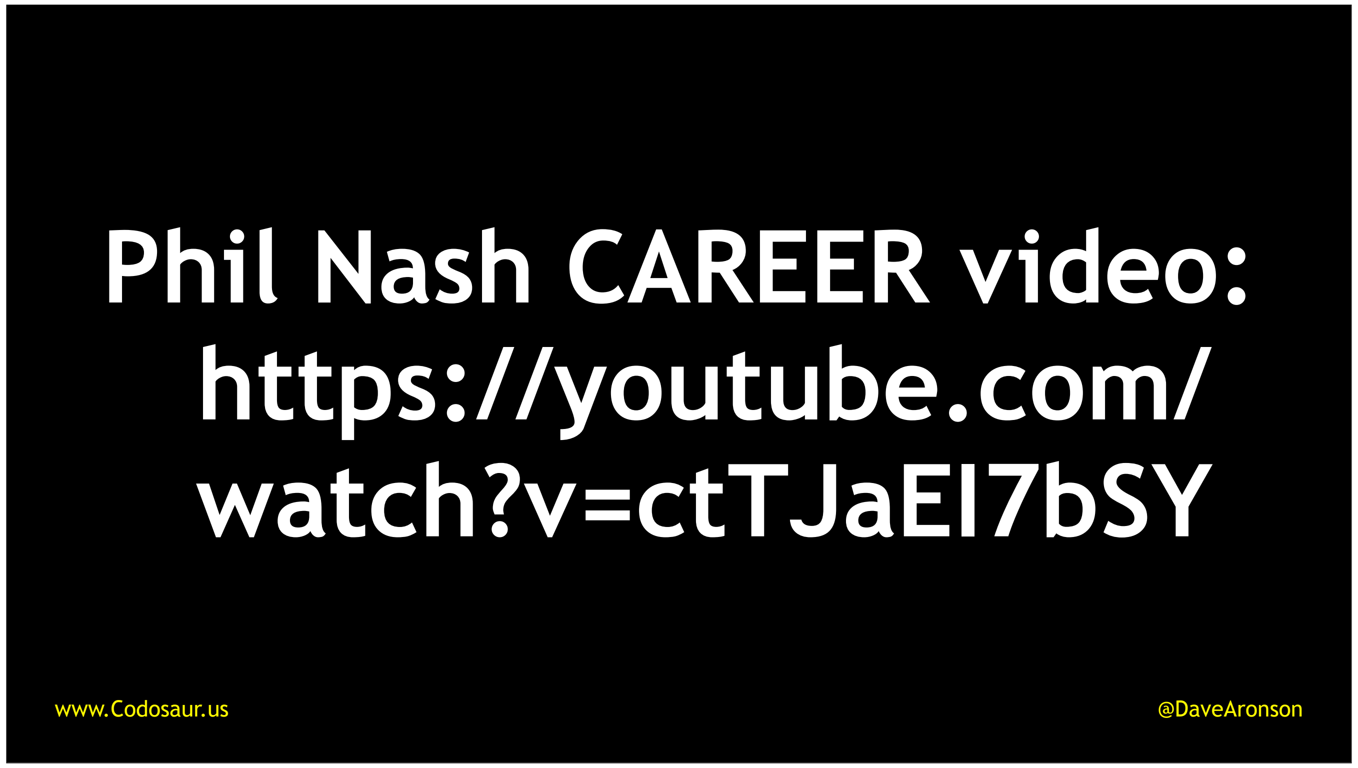
CAREER	=>	ACRUMEN
Correct	=>	Correct
Applicable	=>	Appropriate + Usable
Resilient	=>	Robust
Efficient	=>	Efficient
Evolvable	=>	Part of Maintainable
Reasonable	=>	Part of Maintainable

ACRUMEN	=>	CAREER
Appropriate	=>	Applicable
Correct	=>	Correct
Robust	=>	Resilient
Usable	=>	Part of Applicable
Maintainable	=>	Evolvable + Reasonable
Efficient	=>	Efficient

www.Codosaur.us

@DaveAronson

. . . he split a few things apart that I kept together, and vice-versa, as shown by these mappings here. Of course there's also the acronym itself. I'll admit that his has the advantage of being a common, modern, English word, with no excess letters, and even *relevant* to most people who would be using the definition! However, I still like mine, because it doesn't repeat any letters, and prioritizes the aspects. Also, . . .



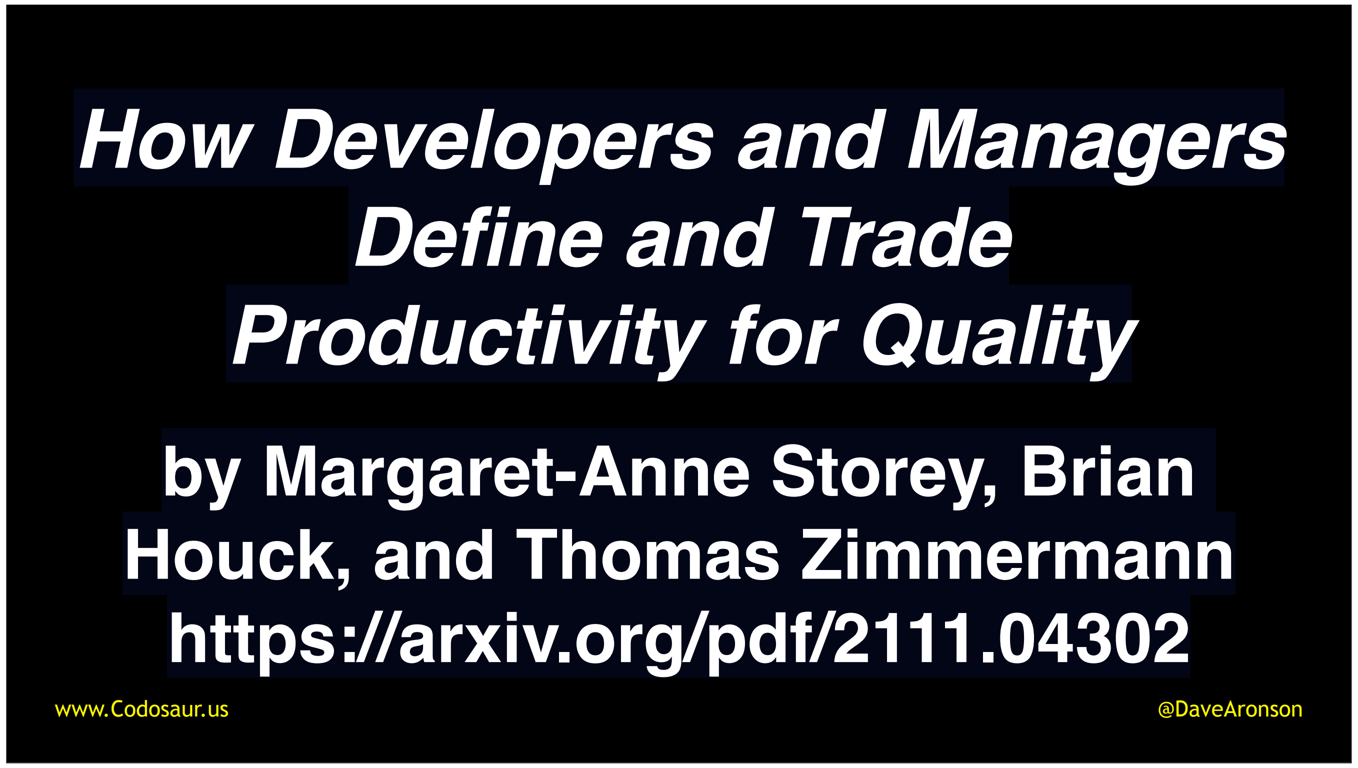
Phil Nash CAREER video: https://youtube.com/watch?v=ctTJaEI7bSY

www.Codosaur.us

@DaveAronson

. . . I would still recommend you watch some of his videos, even after you know all about ACRUMEN, and there's the link to one of them. I say that because when *he* speaks about it, he makes some great additional points. He tends to focus on the *interplay between* aspects, like how some pairings are opposed while others are more synergistic and some are orthogonal, versus my usual deep dives on each one *separately*. That did however inspire thirty new subchapters in the book I've been (very slowly) writing about ACRUMEN.

Another interesting one came from . . .



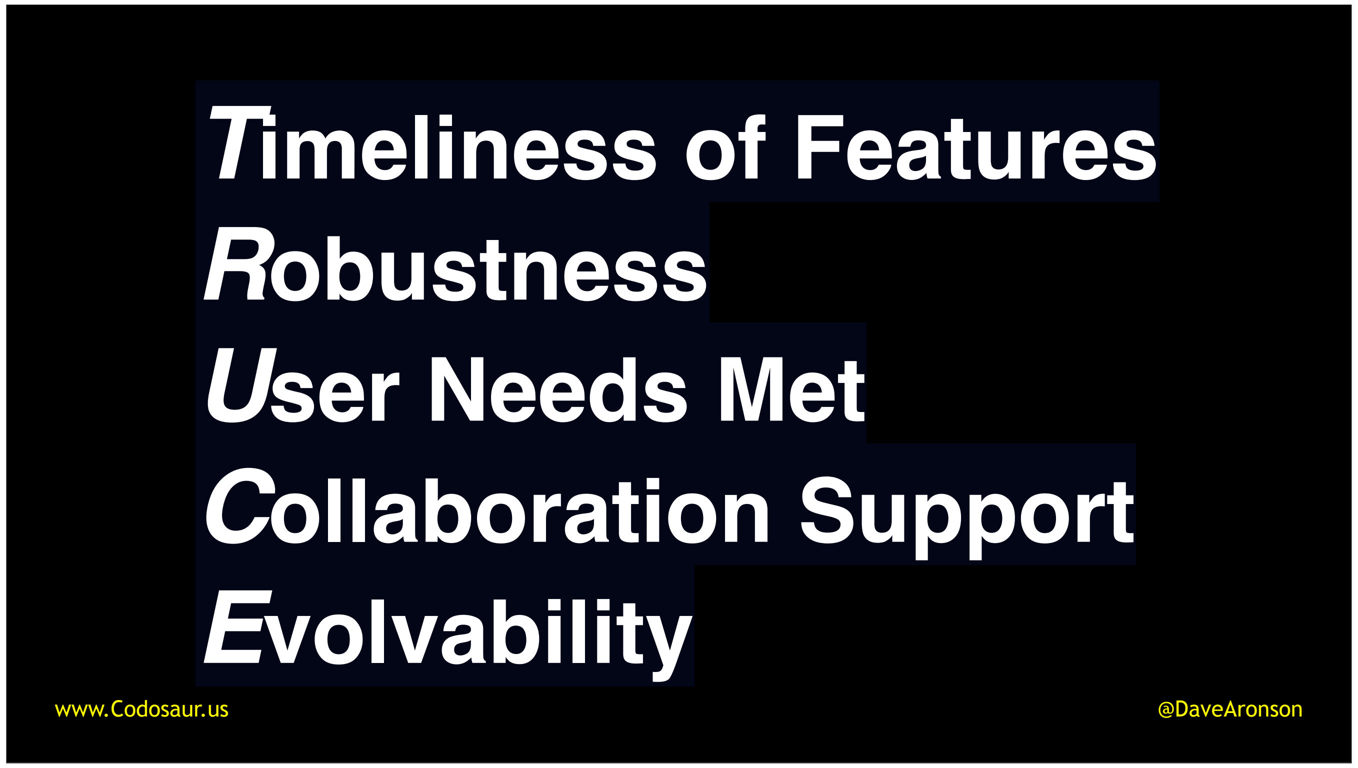
How Developers and Managers Define and Trade Productivity for Quality

by Margaret-Anne Storey, Brian Houck, and Thomas Zimmermann
https://arxiv.org/pdf/2111.04302

www.Codosaur.us

@DaveAronson

. . . the paper “How Developers and Managers Define and Trade Productivity for Quality”, by Margaret-Anne Storey, Brian Houck, and Thomas Zimmermann. The main bit of interest to me was how this paper examines the difference in how developers and managers, separately and differently, define productivity and quality. It winds up defining quality with the acronym . . .

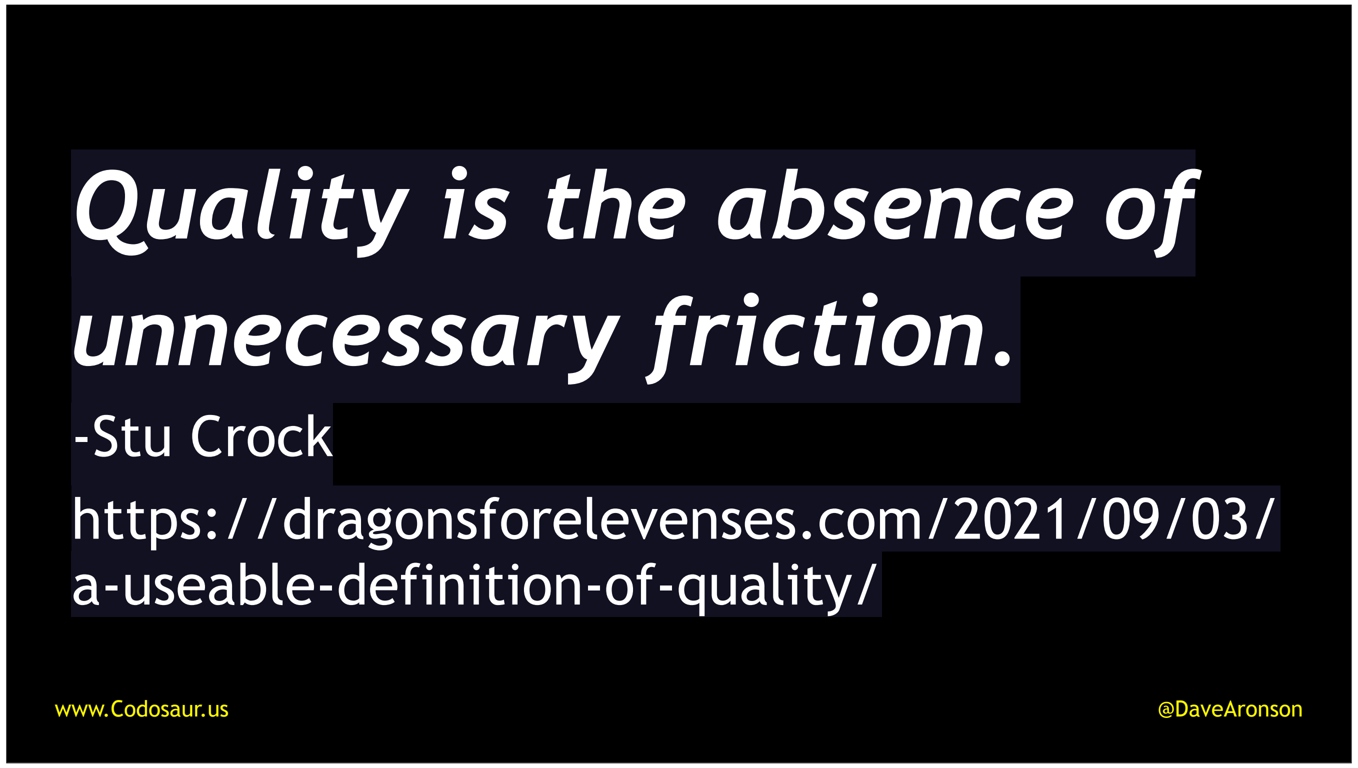


Timeliness of Features
Robustness
User Needs Met
Collaboration Support
Evolvability

www.Codosaur.us @DaveAronson

. . . TRUCE, which as you can see here, stands for Timeliness of Features, Robustness, User Needs met, Collaboration Support, and Evolvability. I have a few nitpicks with how this one is organized, but it's still worth considering, especially for *their* purpose, of comparing the *difference* in the two *groups'* definitions, rather than a more general definition like I was after. Mainly, I didn't want to take into account exactly when it was created, especially individual features, so as to evaluate timeliness, focusing instead on its quality *now*, and this one covers things like Correctness and Efficiency only implicitly, as one might consider them parts of User Needs Met. So, going through the TRUCE words one by one: Timeliness just didn't fit what I was looking for. Robustness, however, is exactly a word I also used! User Needs Met is most of Appropriateness, just not addressing the other categories of stakeholders. Collaboration Support basically means it's maintainable specifically in ways that support developer collaboration on it, like modularity, documentation of the design and implementation, and so on. So, that's a decent chunk of what I call Maintainability, with some further chunks provided by Evolvability. So all in all, pretty good!

Lastly, Stu Crock has a very interesting and very *short* definition:



Quality is the absence of unnecessary friction.

-Stu Crock

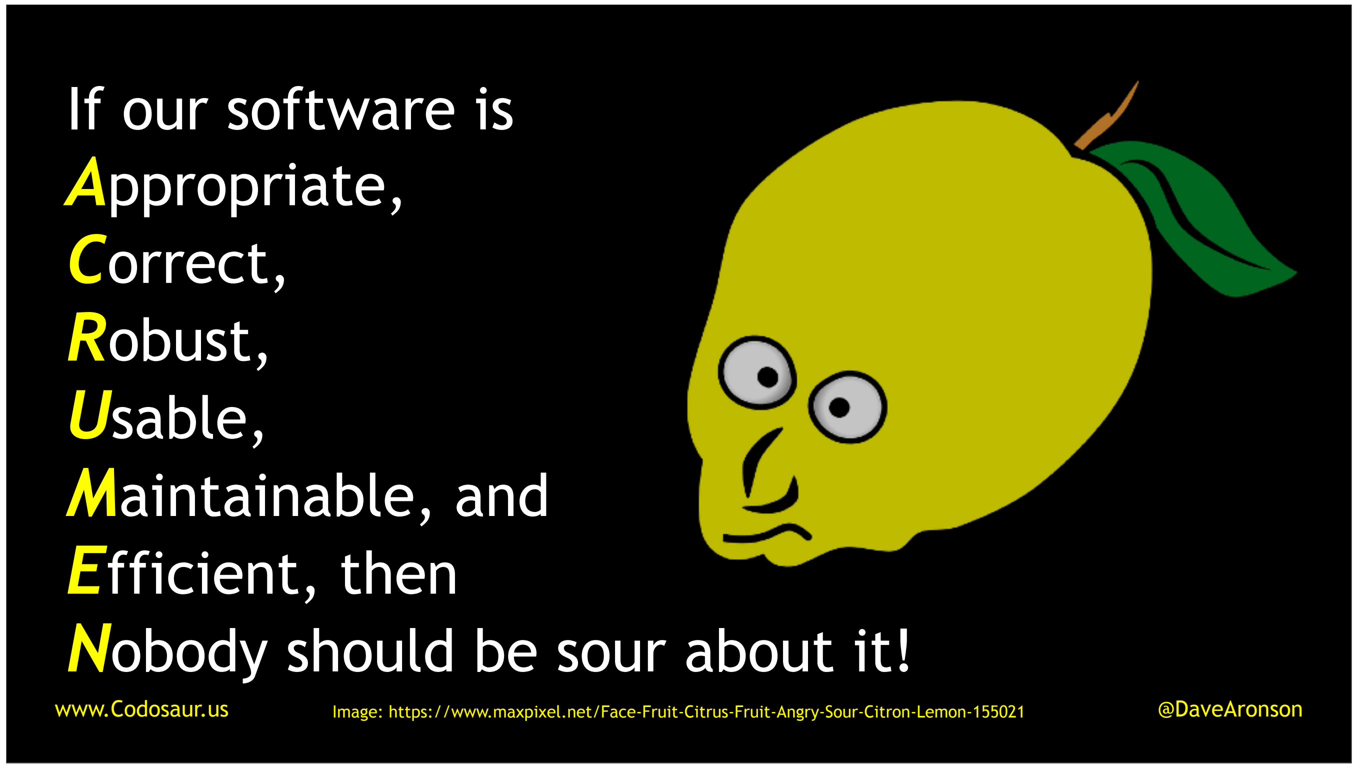
https://dragonsforelevenses.com/2021/09/03/a-useable-definition-of-quality/

www.Codosaur.us

@DaveAronson

“the absence of unnecessary friction”. This wasn’t at all the *sort* of definition I was looking for, but it *is* a wonderful way to spark some very useful discussions about sources of friction in our systems, and to what extent some of that friction may actually be *necessary*. Even the underlying idea that some friction may sometimes be necessary, is an enlightening thought, and he provides some examples.

In conclusion . . .



If our software is
Appropriate,
Correct,
Robust,
Usable,
Maintainable, and
Efficient, then
Nobody should be sour about it!

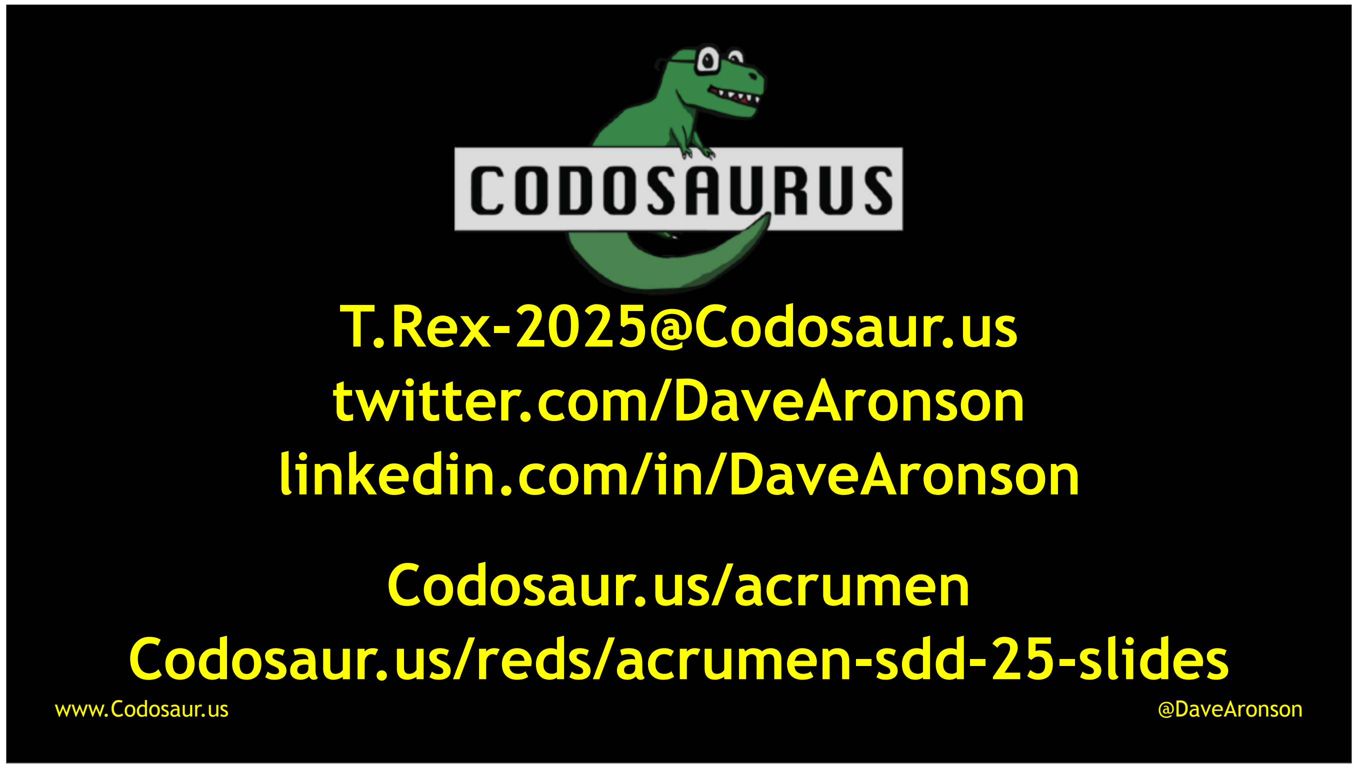
www.Codosaur.us

Image: https://www.maxpixel.net/Face-Fruit-Citrus-Fruit-Angry-Sour-Citron-Lemon-155021

@DaveAronson

. . . if we software developers remember to do what we can to make sure that our software is Appropriate, Correct, Robust, Usable, Maintainable, and Efficient, then nobody should have any cause to be SOUR about the FRUITS of our labors.

If you have any questions, I'll take some now, we have about ??? minutes left, plus I'll be around for the rest of the conference, and if you think of something later . . .


T.Rex-2025@Codosaur.us
twitter.com/DaveAronson
linkedin.com/in/DaveAronson

Codosaur.us/acrumen
Codosaur.us/reds/acrumen-sdd-25-slides

www.Codosaur.us @DaveAronson

. . . there's my contact info, plus the URLs for my web page about ACRUMEN, and for this whole slide deck, complete with a full script, which I've *mostly* stuck to. Any questions?

AFTER QUESTIONS: Please remember to vote about the talks, and if you don't give me the top rating, for whatever reason, I only ask that you *tell me* what should be improved, and ideally how. Now let's all get out there and *write better software*, now that we know what that even means!