ACRUMEN:

What IS Software Quality, Anyway?

by Dave Aronson
T.Rex-2023@Codosaur.us

codosaur.us/reds/acrumen-front-23-slides

(NOTE TO SELF: HAVE BUSINESS CARD READY!)

Current time: ~20 mins — want ~20 (+5 for Q&A), so WATCH THE AD-LIBS!

ACRUMEN:

What IS Software Quality, Anyway?

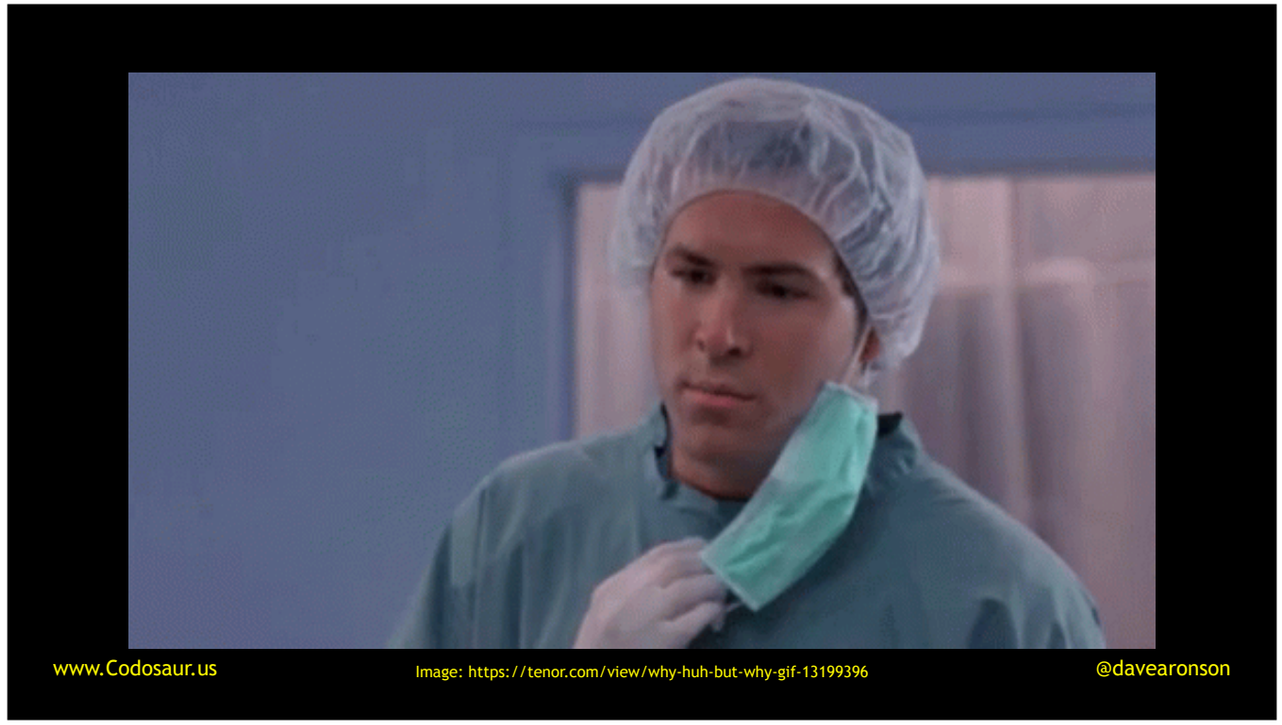
by Dave Aronson
T.Rex-2023@Codosaur.us

codosaur.us/reds/acrumen-front-23-slides

www.Codosaur.us

@davearonson

Hello, Arlington! I'm Dave Aronson, the T. Rex of Codosaurus, and I’m here to tell you about my definition of software quality. But, . . .



. . . why? Quality is something we all agree we need, but if we don't agree on a definition, it's very hard to achieve, or at least to get someone else to acknowledge that we have. Several years ago, I was *looking* for a good definition, but all the ones I found had serious problems. 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. That's fine for talking amongst *ourselves*, but I wanted a definition that *even non-technical* people would understand, so they could understand our *work* better, and give us more precise *feedback* about *exactly how* our software sucks. (And you know most of it does.)

Some definitions were . . .

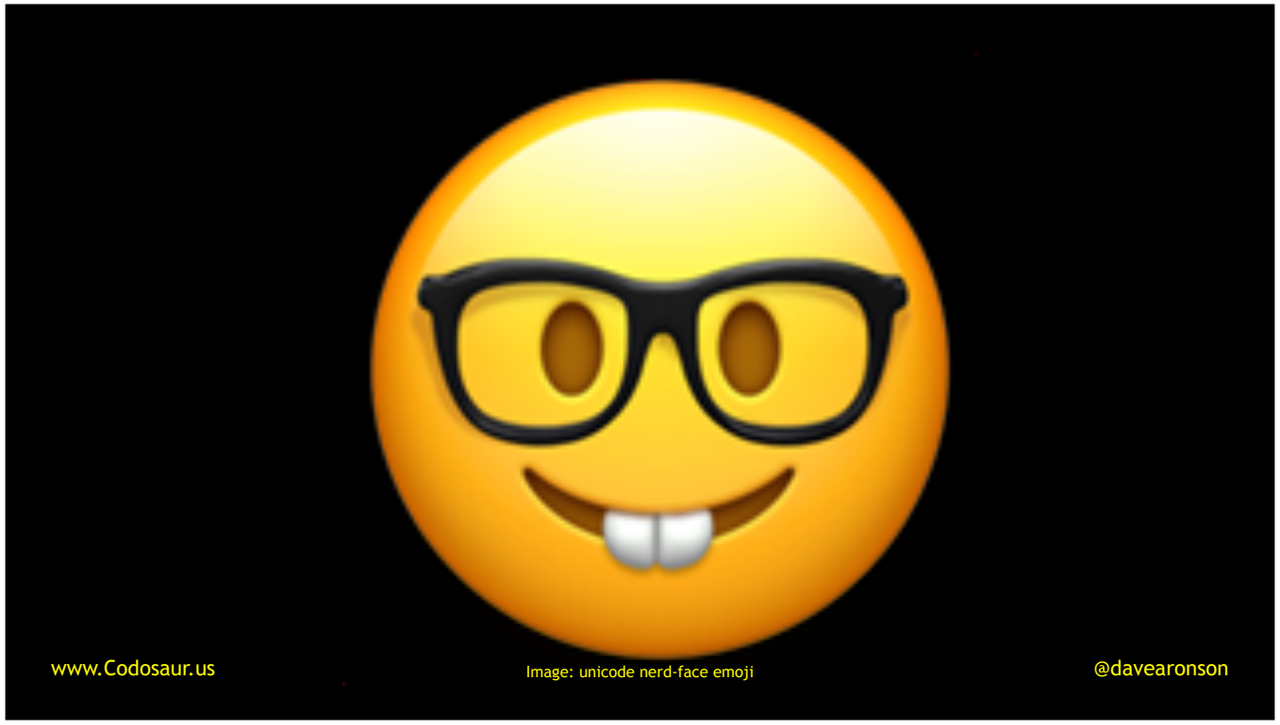
www.Codosaur.us

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

@davearonson

. . . proprietary, requiring us to buy expensive software or documents. Some were only applicable within the context of certain styles or technologies, often also proprietary. I felt that all of that was just plain wrong. I wanted something that *everybody* could use, for *free*.

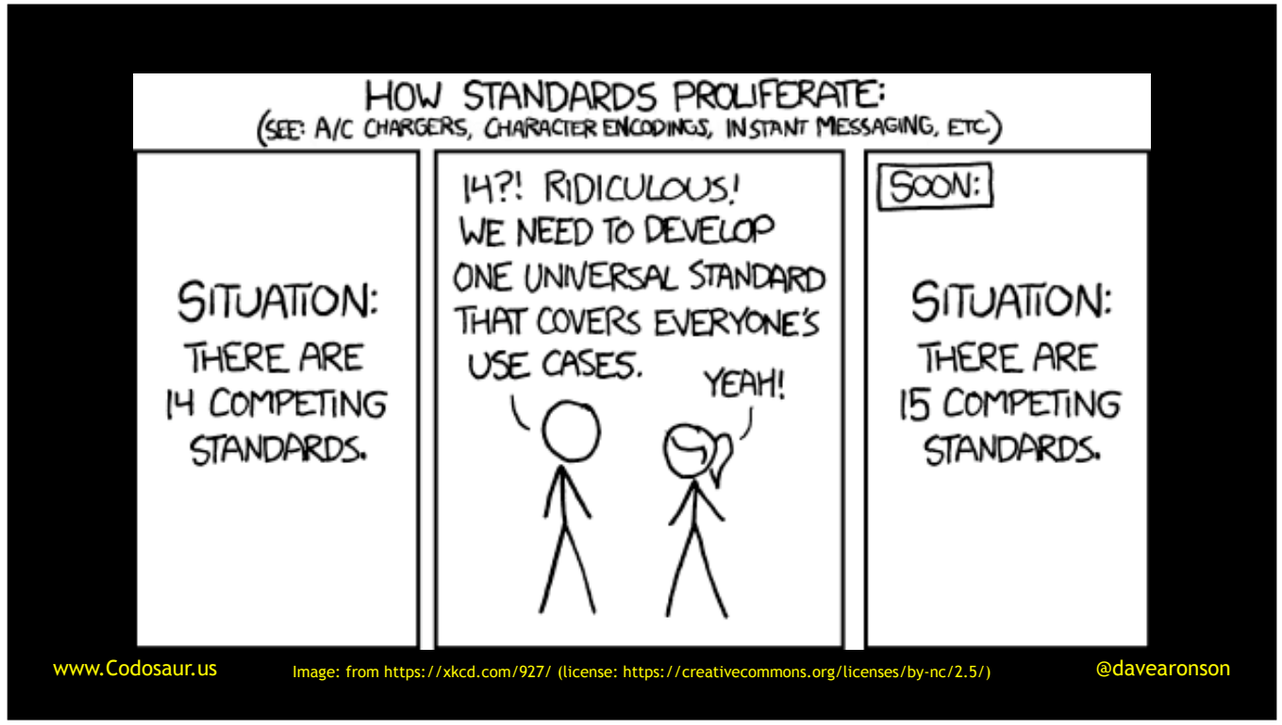
Some definitions focused exclusively on issues of interest to . . .



. . . us developers, ignoring the needs of the users and other stakeholders. Some weren’t even about the software at all, but all about the . . .

. . . process, or the byproducts, like meetings, or issue tracking, or documents. I wanted something more focused on the software itself, and *de*scriptive rather than *pre*scriptive, telling people what software quality *is*, rather than relying on tools and techniques that were likely to change quickly.

I didn't see any that I liked, nor that was commonly accepted, so in the spirit of . . .



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.

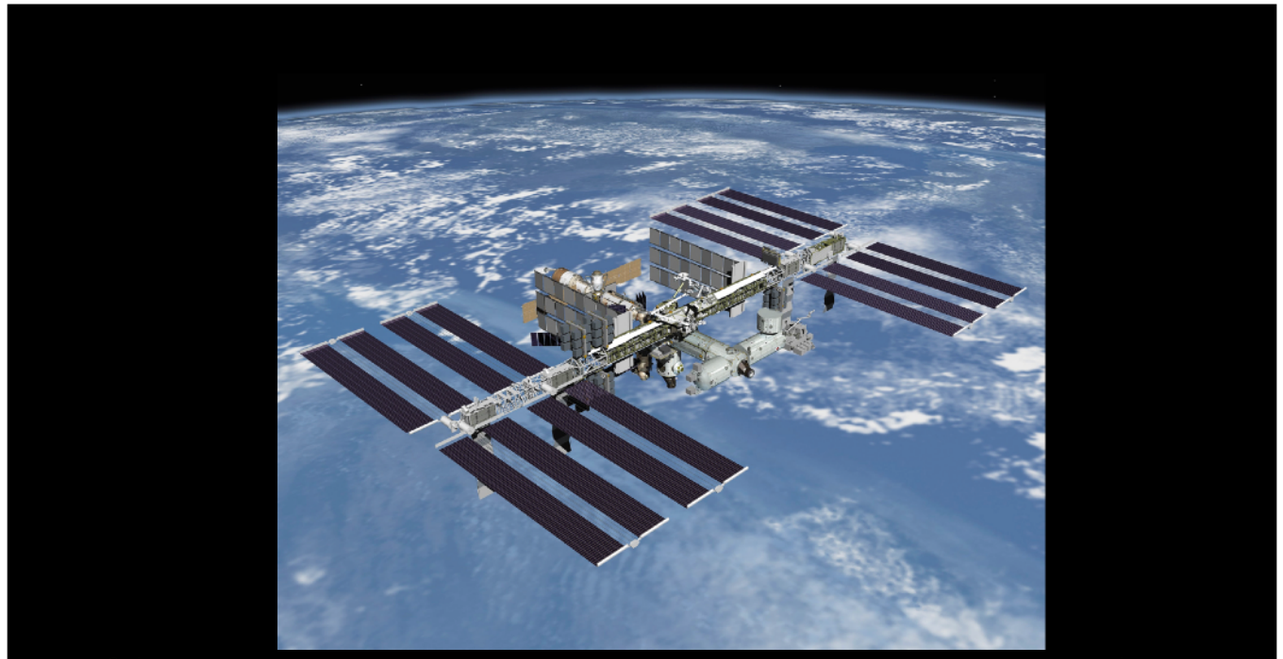
To keep it simple, I (step back) zoomed out from . . .

www.Codosaur.us

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

@davearonson

. . . down in the weeds, where we developers tend to live, 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 just six aspects, with simple names and *relatively* simple explanations. The explanation literally fits on the back of a business card, and (HOLD UP BIZ CARD!) here's mine to prove it. See me afterward if you want one as a cheat-sheet.

I call this list of aspects . . .

ACRUMEN

www.Codosaur.us

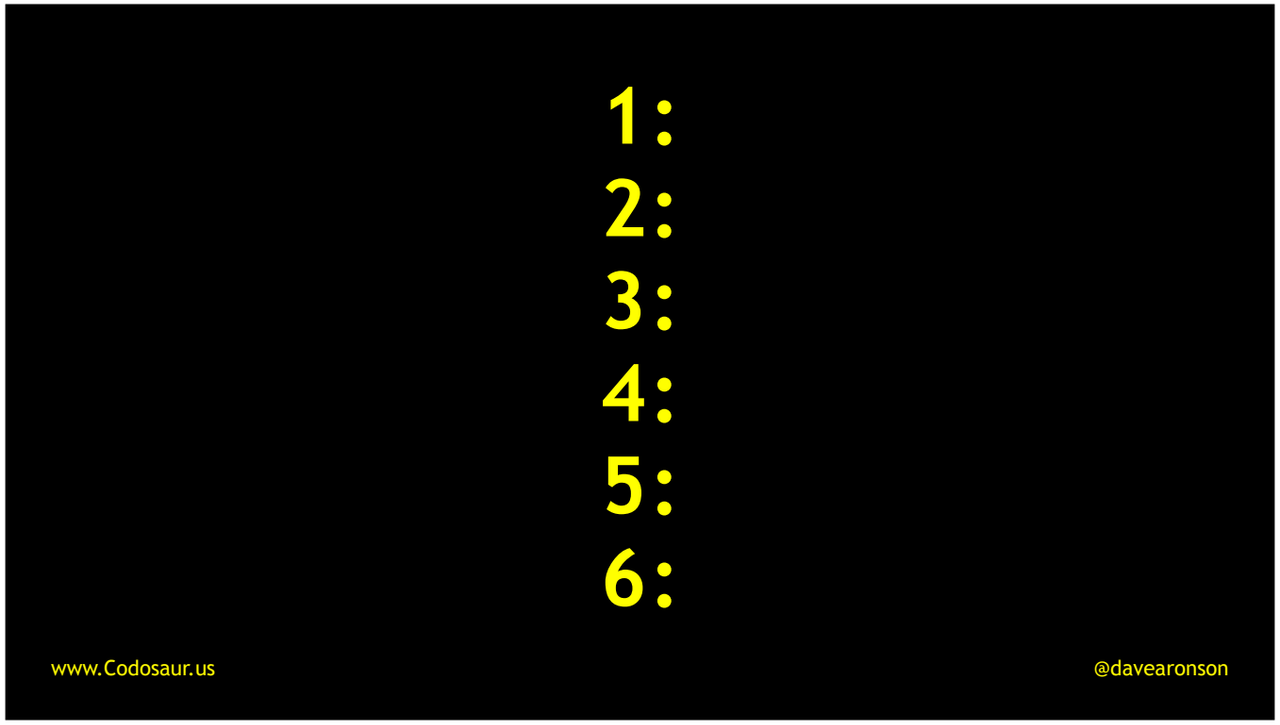
@davearonson

. . . ACRUMEN, but what does that mean? Originally, it was a Latin word, meaning sour fruit, such as . . .

. . . lemons. That's why lemon yellow is the Official Color of ACRUMEN.

But what is ACRUMEN in *this* context?

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



. . . puts them in priority order.

By now you’re probably wondering, SO WHAT ARE THOSE 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:

Appropriate

Correct

Robust

Usable

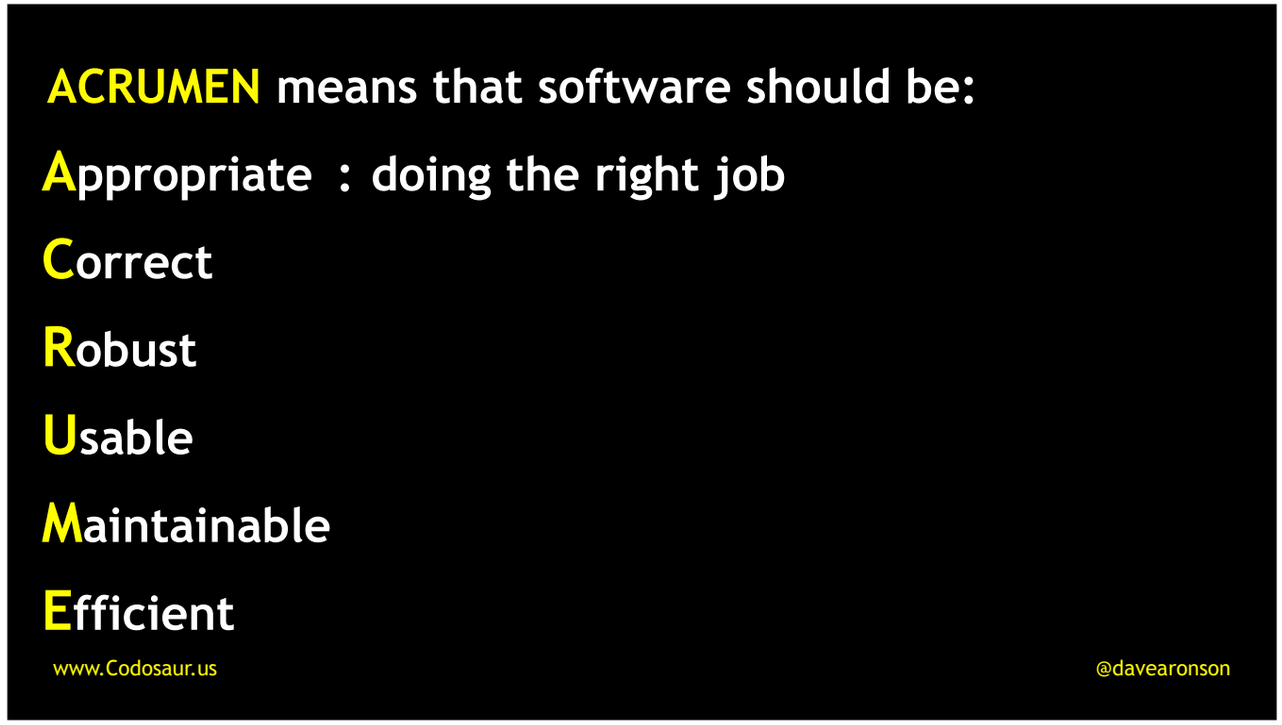
Maintainable

Efficient

www.Codosaur.us

@davearonson

. . . Appropriate, Correct, Robust, Usable, Maintainable, and Efficient. But what does all *that* mean? First, it needs to be . . .



. . . *doing what the stakeholders need* it to do, in other words, doing the *right job*. Then it needs to be . . .

ACRUMEN means that software should be:

Appropriate : doing the right job

Correct : doing the job right

Robust

Usable

Maintainable

Efficient

www.Codosaur.us @davearonson

. . . *doing* that job correctly, or in other words, doing the *job right*. It should be . . .

ACRUMEN means that software should be:

Appropriate : doing the right job

Correct : doing the job right

Robust : hard to make malfunction *or seem to*

Usable

Maintainable

Efficient

www.Codosaur.us

@davearonson

. . . hard for anyone to make it malfunction, or even seem to, but it should be . . .

ACRUMEN means that software should be:

Appropriate : doing the right job

Correct : doing the job right

Robust : hard to make malfunction *or seem to*

Usable : easy for users to use

Maintainable

Efficient

www.Codosaur.us

@davearonson

. . . easy for the users to use and . . .

ACRUMEN means that software should be:

Appropriate : doing the right job

Correct : doing the job right

Robust : hard to make malfunction *or seem to*

Usable : easy for users to use

Maintainable : easy for devs to change

Efficient

www.Codosaur.us @davearonson

. . . for the developers to change. Last, *dead last* despite how we developers tend to worship this, it should be . . .

ACRUMEN means that software should be:

Appropriate : doing the right job

Correct : doing the job right

Robust : hard to make malfunction *or seem to*

Usable : easy for users to use

Maintainable : easy for devs to change

Efficient : going easy on resources

www.Codosaur.us @davearonson

. . . easy on resources.

So, what does the N stand for? Nnnnn . . .

. . . nothing! I just tacked it on to make a real word.

While all that’s fresh in our minds, I'll address some . . .

. . . frequently asked questions. The first is, aside from going into detail on the tips, how do we actually *use* ACRUMEN itself, the list of aspects?

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

. . . *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, be it on a scale of 1 to 10, or by simple triage, or is it *good enough* for *our needs?* And if it's not good enough, what can be done to *. . .*

www.Codosaur.us

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

@davearonson

. . . *make* it so?

In the short term, we can ensure that our *projects* are likely to *meet* these criteria. In the long term, we can ensure that our *processes support* these criteria, by including helpful activities, maybe even an *explicit evaluation against* these criteria. We can also set . . .

www.Codosaur.us

Image: https://www.flickr.com/photos/bensutherland/205606714

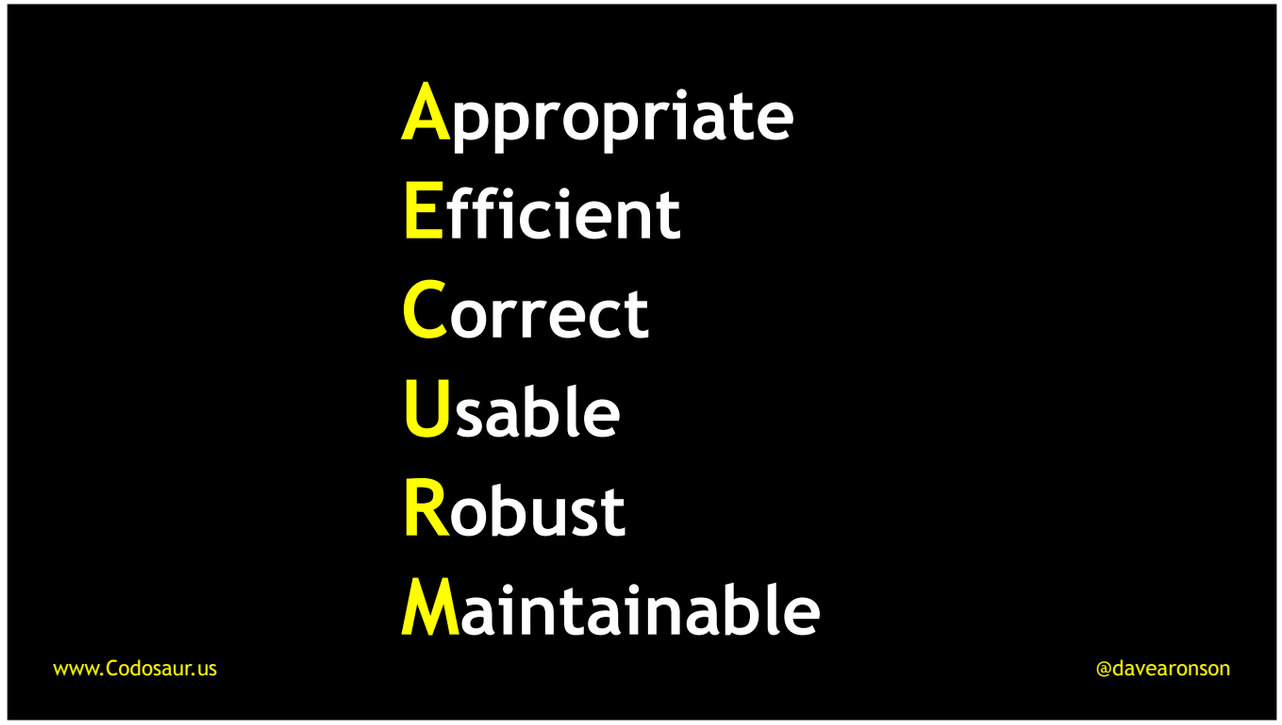
@davearonson

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

Another frequently asked question is: . . .

. . . is ACRUMEN, or rather ACRUME, always the right order?

The answer is, no, ACRUME is just the *typical* case. Your mileage may well vary. Consider a company-internal command-line physics simulation tool, using a standard algorithm. I'll spare you the reasoning, but its list may well look more like . . .



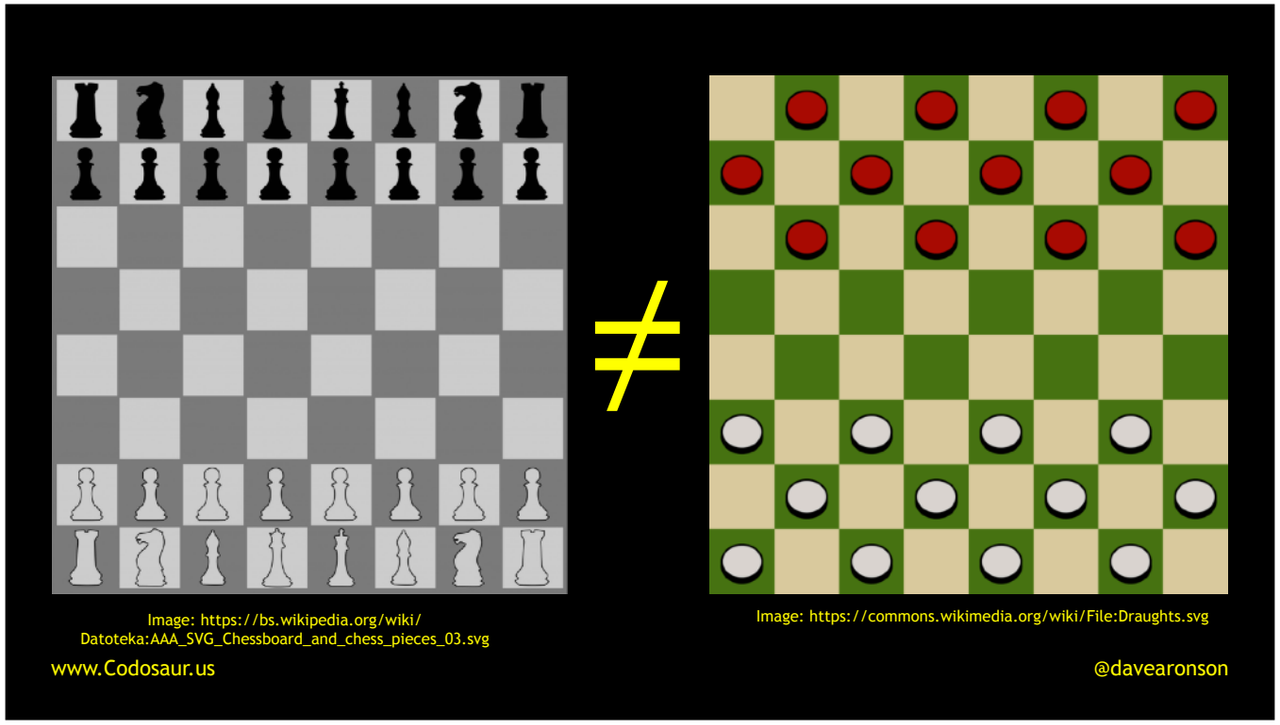
. . . this, AECURM, rather than ACRUME. The only real constant is that Appropriate will always be at the top. We'll soon see why, because now we'll look closer at each aspect, and up first is of course . . .

. . . 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 . . .

. . . *all* the others *put together*. 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 Appropriate, let's try a little thought experiment. Suppose you want a program to play . . .



. . . *checkers*, and I write for you the world's greatest *chess* playing program. It's as correct, robust, usable, maintainable, and efficient as anyone could ever want. But you probably won't be happy with it, because . . . it's not checkers. It's not what you asked for. It's not what you *need*. In ACRUMEN terms, it's not *appropriate*.

So how do we achieve appropriateness? In an ideal world, we would have . . .

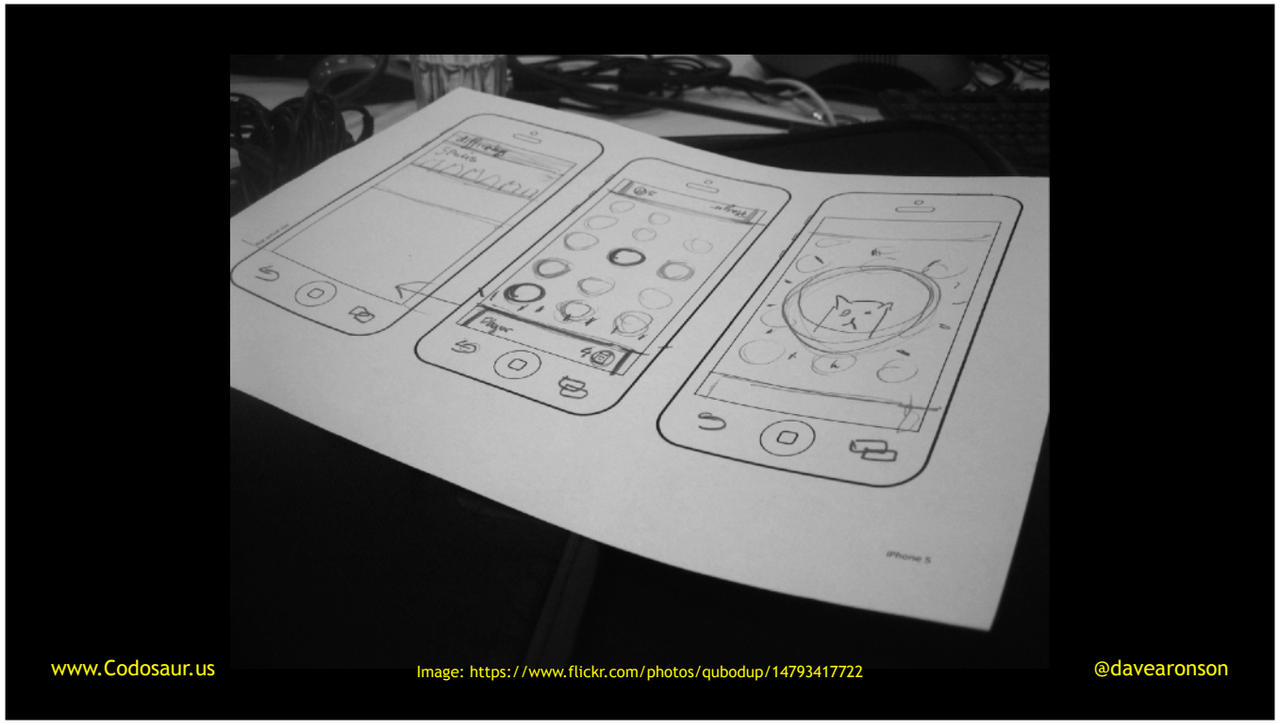
www.Codosaur.us

Image: https://www.flickr.com/photos/wocintechchat/22543243101

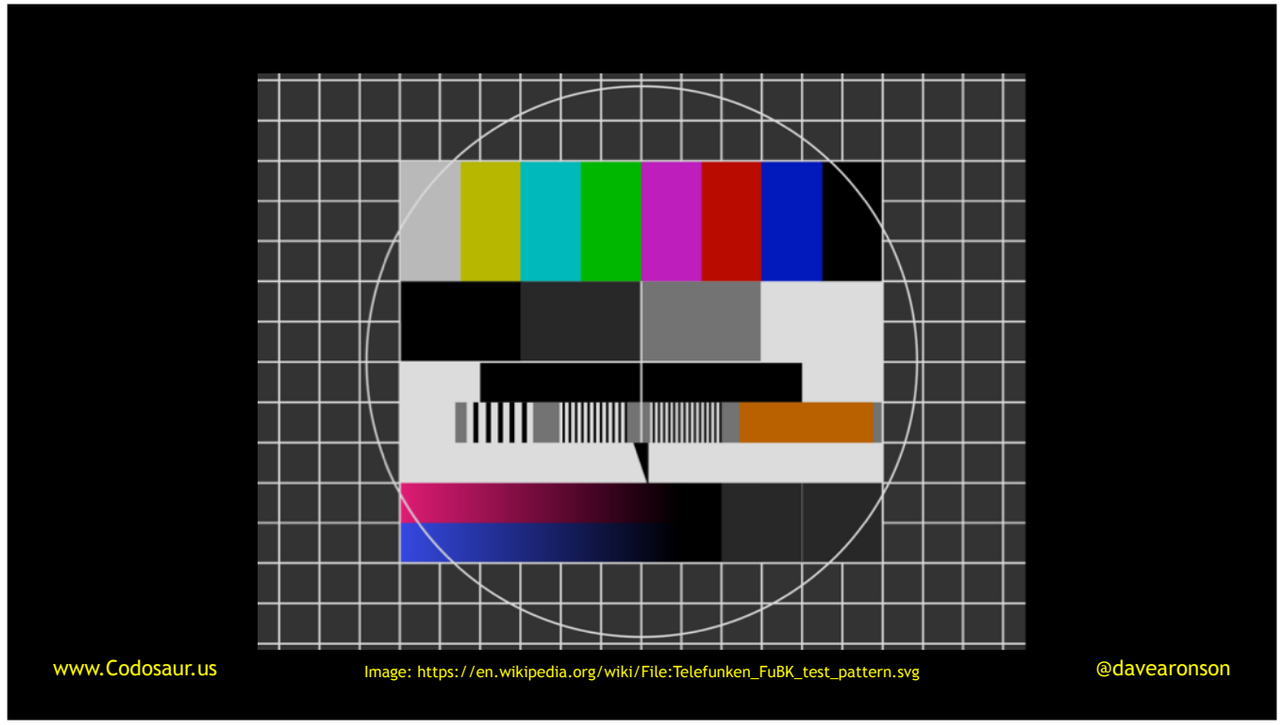
@davearonson

. . . frequent direct contact with the stakeholders, to ask them questions and get their feedback. Unfortunately, we don't usually get that opportunity, and often not even requirements analysts, or business analysts. So we usually have to settle for occasional remote indirect contact with a representative of some stakeholders. It doesn't work quite as well, but having *some* communication with *someone* with a clue, is *vital*.

Once we think we have a good grasp of the stakeholders' needs, we can show them, or their representatives, . . .



. . . mockups and prototypes of what we *intend* to do, and demos of what we *have* done. This gives them a chance to *correct our wrong ideas* of their needs, before we go too far down the wrong rabbit-hole. There's another thing, though, that I'll be returning to over and over in this talk. We can propose . . .



. . . *tests!* In particular, I recommend the Given/When/Then pattern: given these preconditions, when this happens, then this is the result. This makes a great link between the worlds of business and tech, because the business people can understand it (especially for user-visible things like most front-end work), and we can turn it into a runnable test.

Our next aspect is . . .

. . . correctness. Nothing can actually *stop* us from *writing* code that isn't correct, at least with the tools we have today. So, the big question is: like the . . .

www.Codosaur.us

Image: https://zh.wikipedia.org/wiki/File:Thermoskanne(hoch,_silber).JPG

@davearonson

. . . Thermos that keeps hot things hot and cold things cold, how do we *know*? (PAUSE!)

As you probably know . . .

www.Codosaur.us

Image: https://commons.wikimedia.org/wiki/File:Test_type_to_test_eyesight,_England,_1920-1960_Wellcome_L0058205.jpg

@davearonson

. . . *tests* let us know whether or not our code is correct . . . *assuming* of course that the tests *themselves* are correct, but that's another story.

I'll skip over a lot of common advice about how many of what kinds of tests to write and how and when, but I'l point out that the usual types of tests, like unit, integration, feature, system, and so on, can only prove the correctness of cases *we thought* to test. But there are some advanced techniques that can help find cases we didn't think of, such as . . .

www.Codosaur.us

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

@davearonson

. . . property-based testing and . . .

www.Codosaur.us

Image: https://pixabay.com/en/genetic-modification-mutant-mutation-549889/

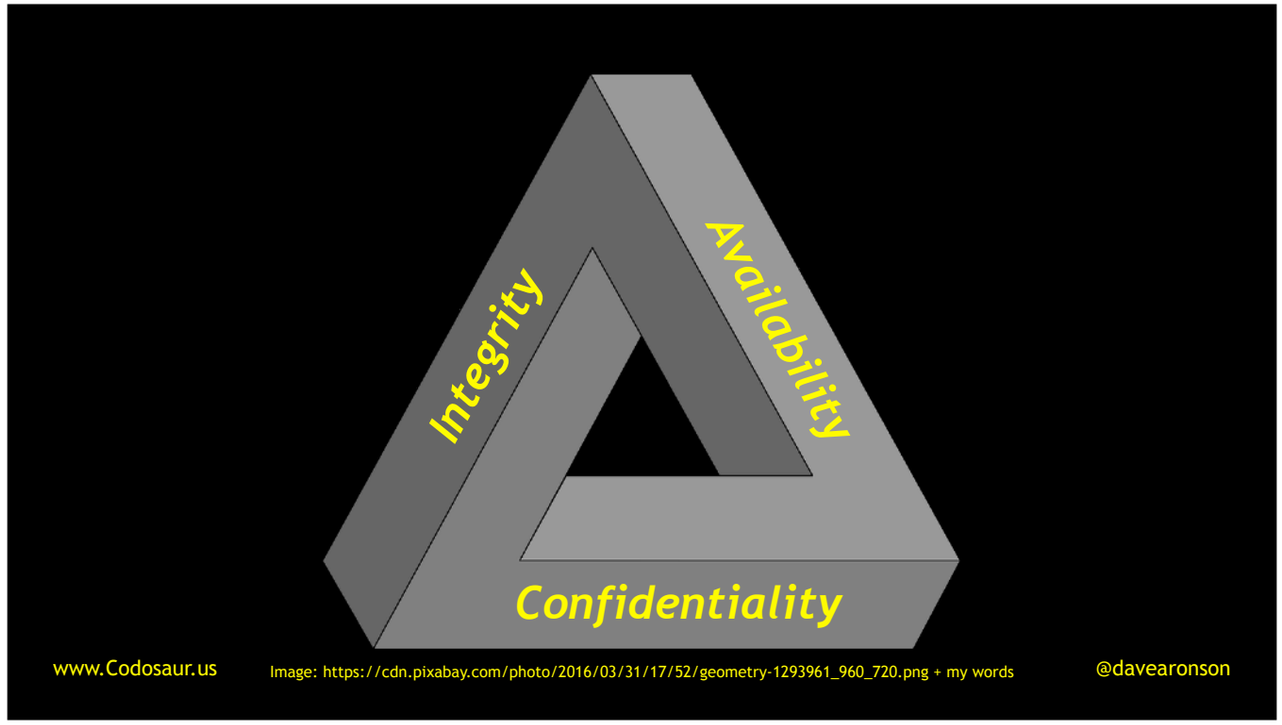
@davearonson

. . . mutation testing. BTW, I also present on mutation testing at conferences, and you can find my videos on Youtube.

We should have enough test coverage, of assorted kinds and levels, and verified to be actually meaningful, to have strong confidence in the correctness of our code.

Next up we have . . .

. . . robustness. 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: . . .



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

www.Codosaur.us

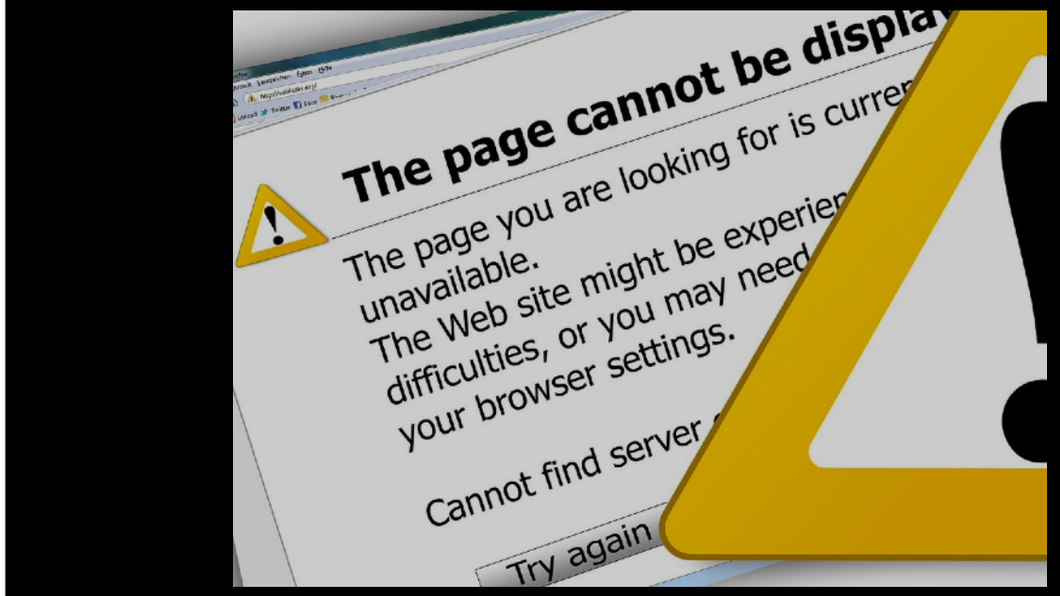
Image: https://commons.wikimedia.org/wiki/File:Somebody_blabbed.jpg

@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, . . .



www.Codosaur.us

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

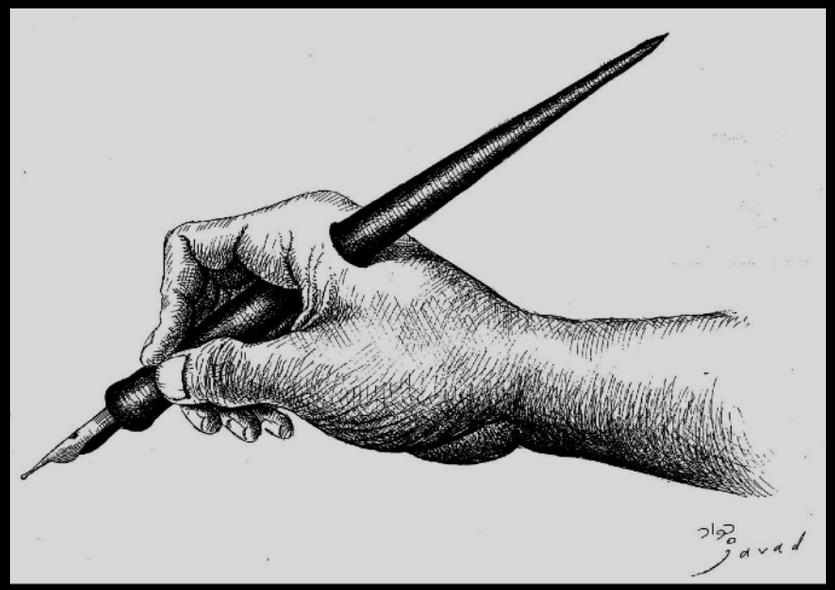
@davearonson

. . . or become unavailable when it’s not supposed to, even if an attacker is trying to . . .

. . . *force* it to do these things.

So how do we achieve all *that?*

Again, we could bring in the experts, which would be . . .



www.Codosaur.us

Image: https://commons.wikimedia.org/wiki/File:Javad_alizadeh_-penetrating-pen.jpg

@davearonson

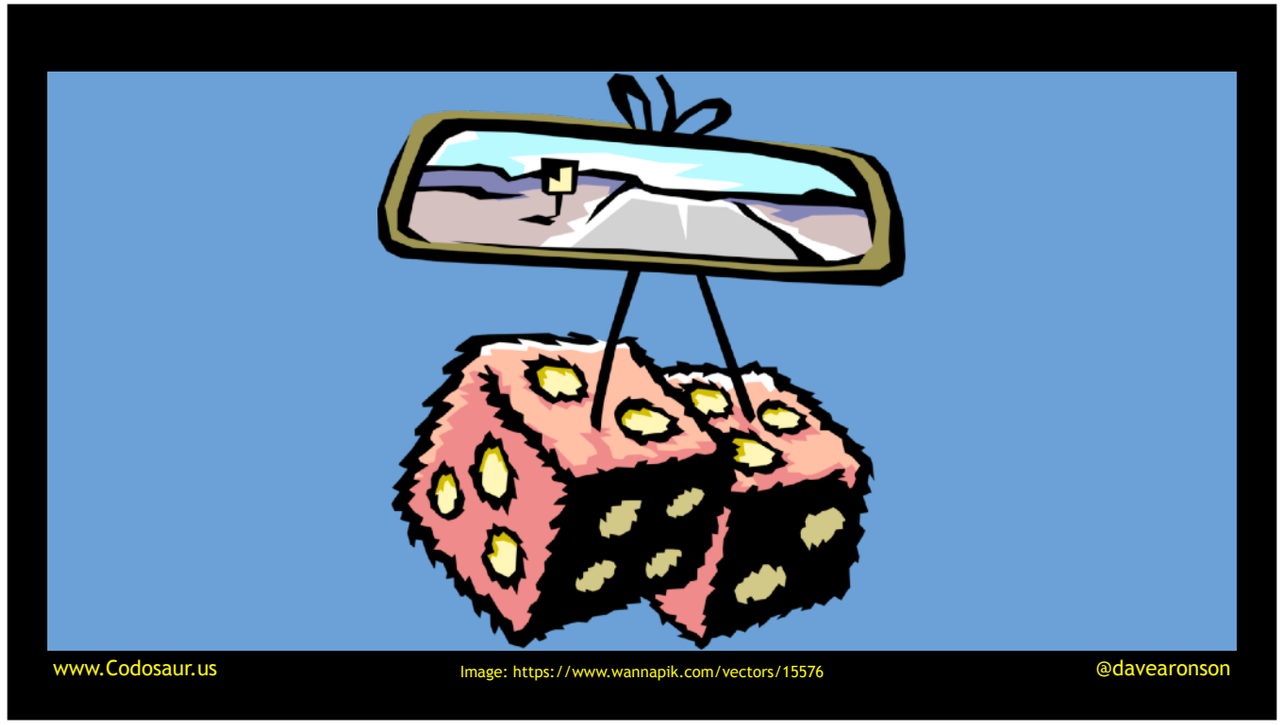
. . . penetration testers, or for short, pen testers. (You can see why I couldn't resist that image!) But, they’re expensive, and disruptive, because they *need* to test the *production* system. So, again, we'll usually have to do without them, but, we can use some of their tools, such as software like . . .

www.Codosaur.us

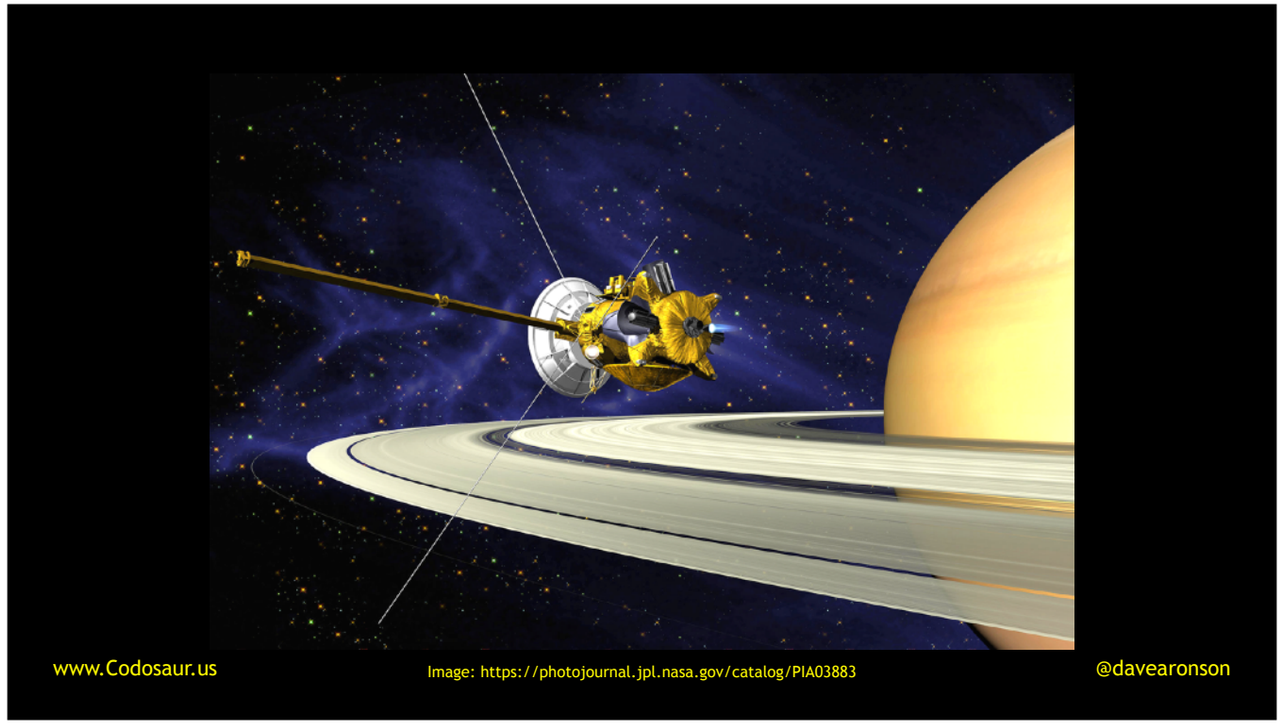
Image: https://pixabay.com/en/girl-child-trampoline-blonde-212022 (CC0)

@davearonson

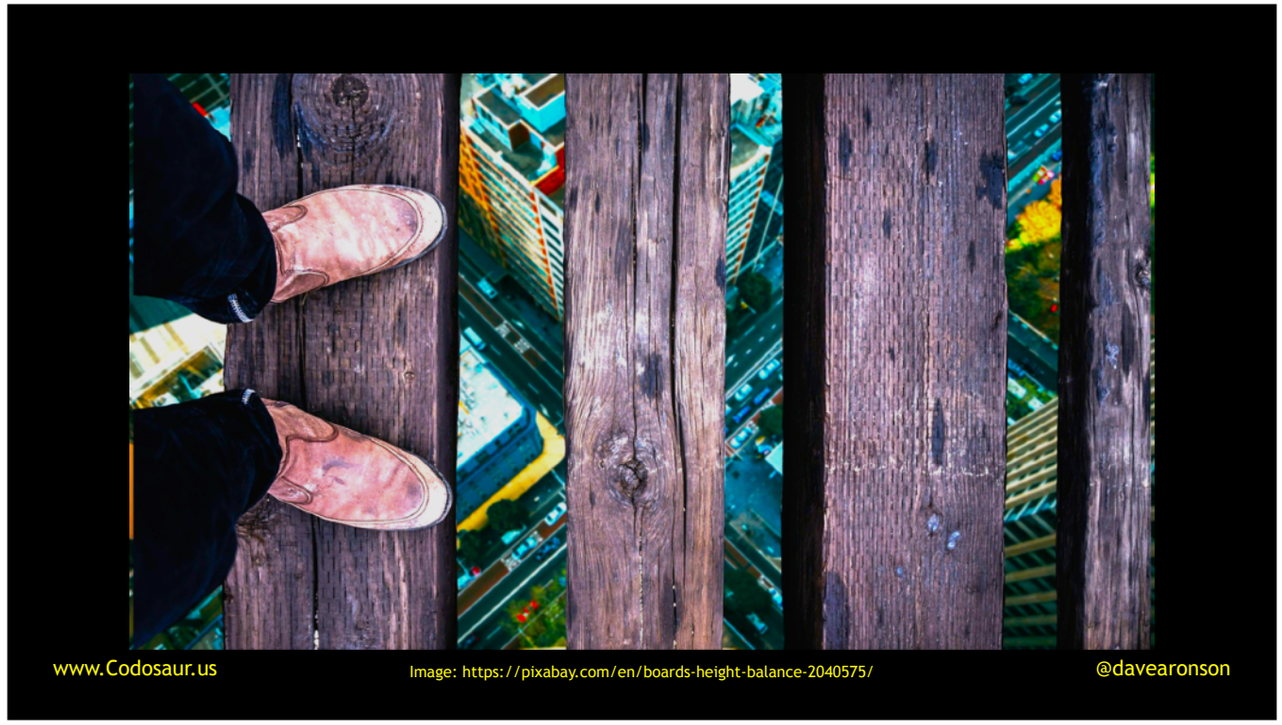
. . . static analyzers, . . .


www.Codosaur.us
Image: https://www.wannapik.com/vectors/15576
@davearonson

. . . fuzzers, and . . .



. . . probes. But that's only covering *actual* fragility. What about *seeming* fragile? For that, we must ask ourselves . . .



. . . what could go wrong. For instance, if the system wants the user to . . .

. . . 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. As implied before, the program should not crash, but also it should not show a mysterious error message like . . .


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

. . . ENOENT . . .

. . . or HTTP Code 500, and a . . .

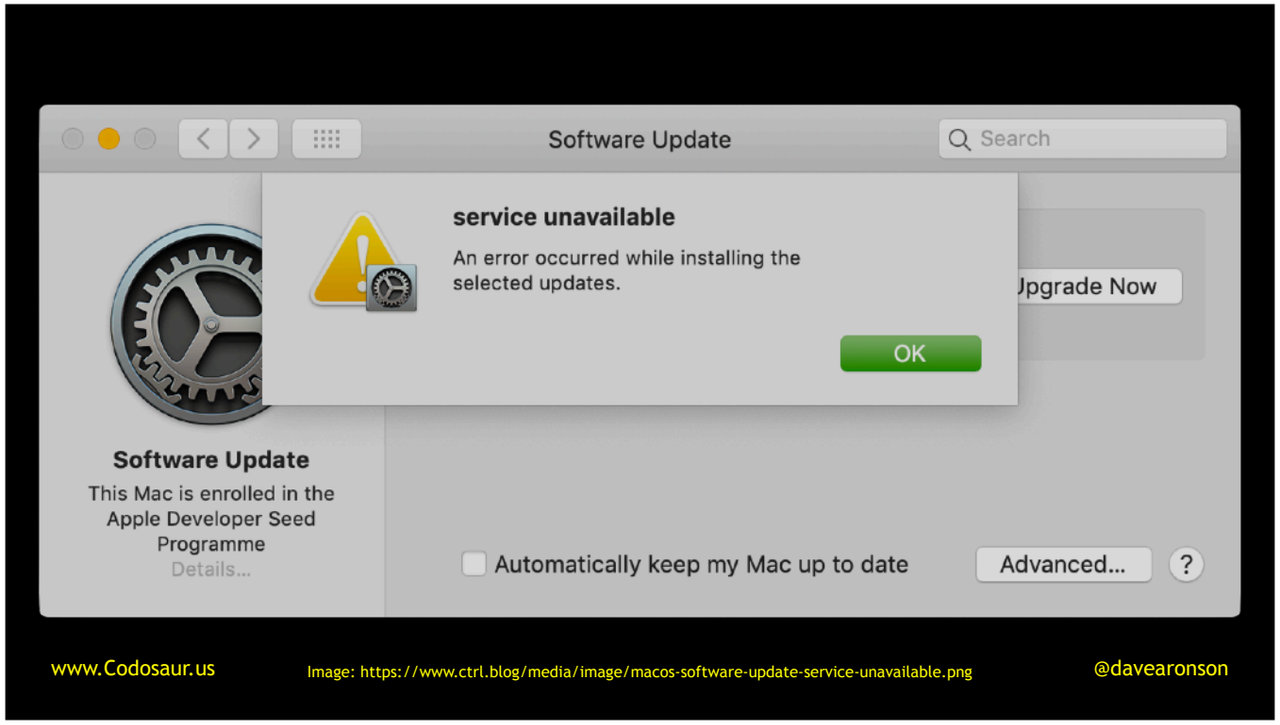
www.Codosaur.us

Image: https://www.flickr.com/photos/artlung/1757819405

@davearonson

. . . stack trace is Right Out!

Instead, it should show . . .



. . . a clear and friendly error message, and let the user try again (if possible).

Our software should handle all *reasonably foreseeable* problems, even external ones like losing a network connection, as gracefully as possible.

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

. . . usability. Hard-to-use software can cause headaches, or even lead the user to do the wrong thing. Remember what happened in Hawai'i in January 2018, due to software that was hard to use? They had a false alarm about an incoming nuclear missile! Just think what could happen if that were our launch system, not just an alarm!

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 be . . .

www.Codosaur.us

Image: https://www.flickr.com/photos/jeepersmedia/13943817487

@davearonson

. . . *easy for everyone* to use, not just *equally difficult*. Granted, the user may be facing various *challenges*. We can *start* with the ones that *accessibility* usually addresses, like low vision, color vision, hearing, or fine motor control. But there are other whole *types* of challenges we should be aware of, like lack of literacy, cultural knowledge, and even *intelligence*. Yes, we may joke about stupid users, but statistically, about half of them *will* be below average. This is not Lake Wobegon Middle School!

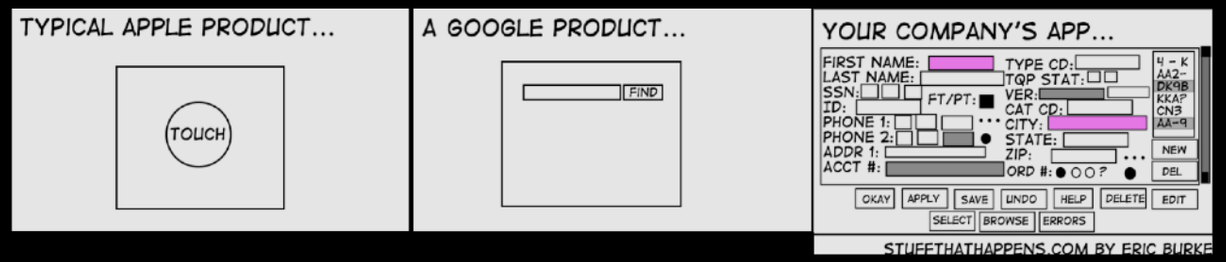
So how do we achieve all this? Once again, ideally we can bring in . . .

www.Codosaur.us

Image: https://commons.wikimedia.org/wiki/File:The_Experts_MET_DT209332.jpg

@davearonson

. . . the experts, like ideally a User Experience expert, maybe a User Interface expert, or at the very least a designer, even an old-fashioned *print* graphic designer. Again, we'll usually have to do without their help, but we can go a long way by applying their principles. 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 a very basic one, the KISS Principle, “Keep It Simple, Stupid”! I think many of us would recognize some of our own work in that cluttered mess at the end.

Also, 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 watch some of our typical users use it (which is what's going on in this photo), and fix their pain points.

The next aspect is the one we usually think of most: . . .

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

. . . maintainability. We'd probably all agree that the basic concept is that "maintainable" software is easy to change. (Thank you, Captain Obvious!) 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 . . .

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

www.Codosaur.us

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

@davearonson

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

www.Codosaur.us

Image: https://commons.wikimedia.org/wiki/File:New_Guy_(5895483627).jpg

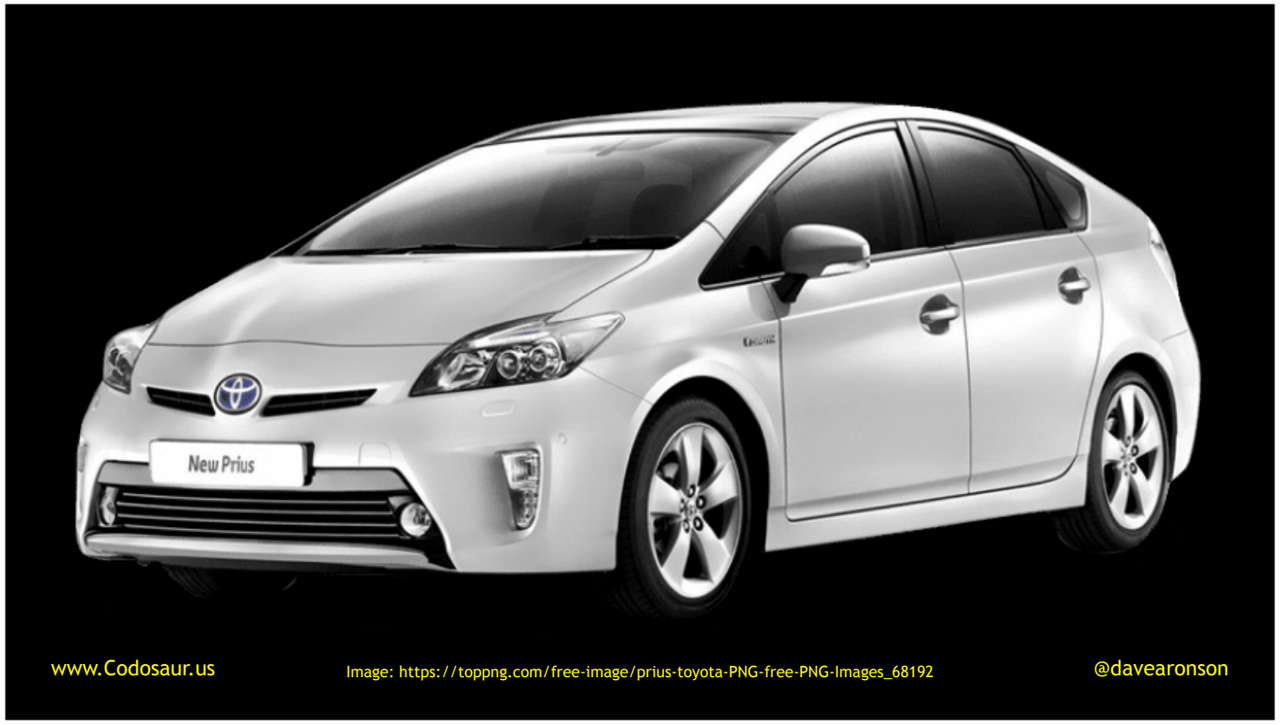
@davearonson

. . . 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 YAGNI and SOLID, and so on, I’m going to stick to my theme and tell you how . . .

. . . testing can help with maintainability. The *old* tests from any *previous* feature additions, bug fixes, and so on, 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 pace with a clear and focused mind, rather than creeping along slowly and erratically because we're terrified of breaking something accidentally and not discovering it until users complain. And *that* speedup is why I mentioned fear at all.

For the final aspect, software should be . . .



. . . efficient, in other words, go easy on resources. Mainly we know about techical resources, like CPU, RAM, bandwidth, and screen space, but there are other kinds, such as the user's *patience* and *brainpower*, and the company's *money!*

So how do we achieve efficiency? Just as there are many kinds of resources, there are many different kinds of inefficiency we could fix, but for now I'm going to focus on fixing the most obvious and common kind: slowness.

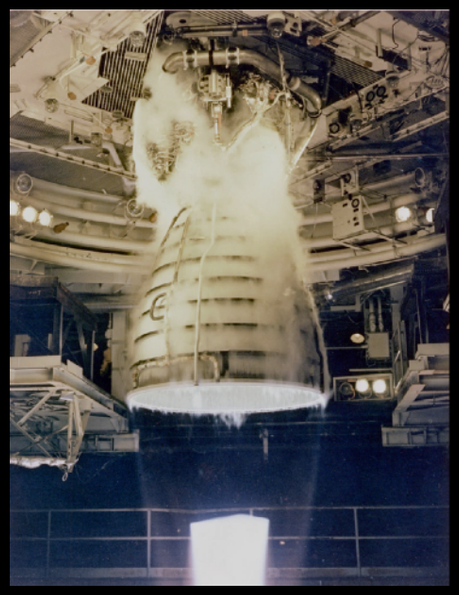
I'm sure we've all had a program run slowly, then we stare at the code, spot where we think it's inefficient, spend a long time optimizing that little piece, run the program again, and . . . it's still slow! Right? Don't do that!

www.Codosaur.us

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

@davearonson

Measure it instead! Humans aren't really very good at spotting the inefficiencies, but there are *profilers* and *packet capture programs* and such, that will tell us *exactly* where, or at least when, we're using too much CPU, RAM, bandwidth, etc. Then we can track down the root cause, fix it, and slap a . . .



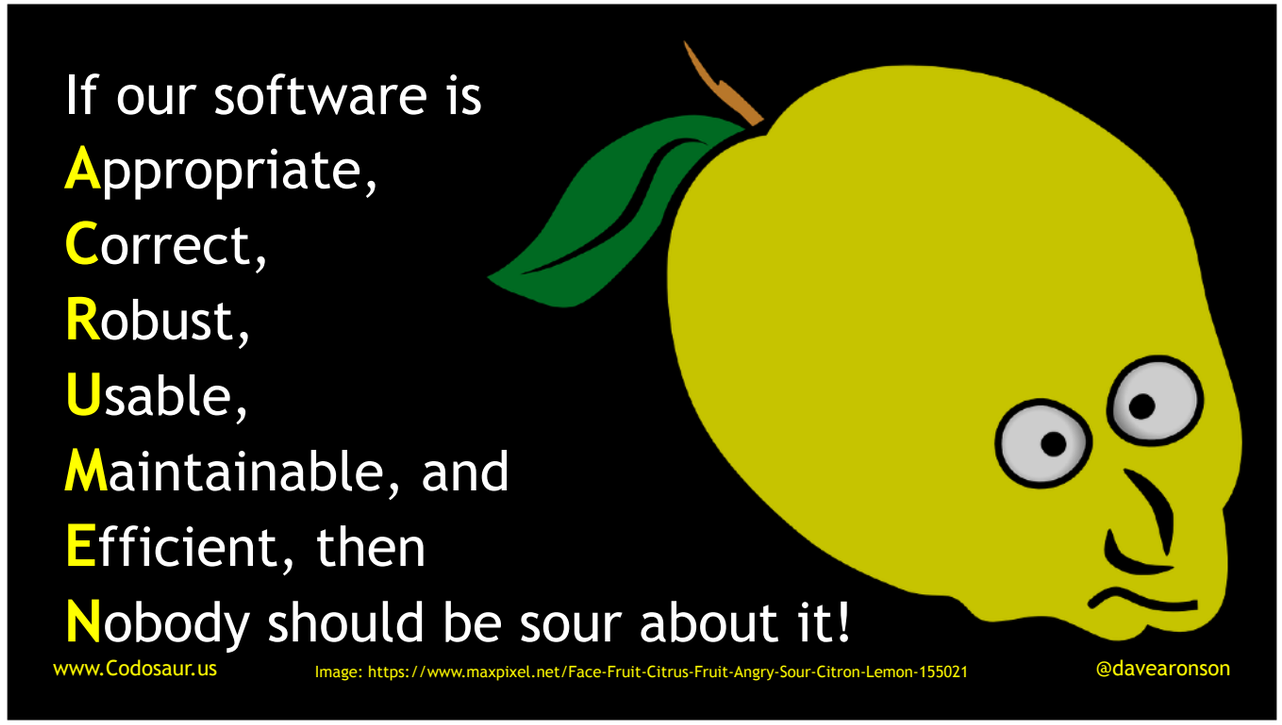
www.Codosaur.us

Image: https://www.nasa.gov/multimedia/imagegallery/image_feature_457.html

@davearonson

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

In conclusion . . .



. . . if we remember 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.

And now . . .

Codosaur.us/acrumen
T.Rex-2023@Codosaur.us
twitter.com/DaveAronson
linkedin.com/in/DaveAronson
toptal.com/#accept-only-candid-coders
codosaur.us/reds/acrumen-front-23-slides

. . . it's your turn! Any questions?