CrossTalk | Aerospace | Aviation

18 Apr 2019 | 19:49 GMT

How the Boeing 737 Max Disaster Looks to a Software Developer

Design shortcuts meant to make a new plane seem like an old, familiar one are to blame

By **Gregory Travis**

The views expressed here are solely those of the author and do not represent positions of IEEE Spectrum *or the IEEE.*

Photo: Jemal Countess/Getty Images

This is part of the wreckage of Ethiopian Airlines Flight ET302, a Boeing 737 Max airliner that crashed on 11 March in Bishoftu, Ethiopia, killing all 157 passengers and crew.

I have been a pilot for 30 years, a software developer for more than 40. I have written extensively about both aviation and software engineering. Now it's time for me to write about both together.

The Boeing 737 Max has been in the news because of two crashes, practically back to back and involving brand new airplanes. In an industry that relies more than anything on the appearance of total control, total safety, these two crashes pose as close to an existential risk as you can get. Though airliner passenger death rates have fallen over the decades, that achievement is no reason for complacency.

The 737 first appeared in 1967, when I was 3 years old. Back then it was a smallish aircraft with smallish engines and relatively simple systems. Airlines (especially Southwest) loved it because of its simplicity, reliability, and flexibility. Not to mention the fact that it could be flown by a two-person cockpit crew—as opposed to the three or four of previous airliners—which made it a significant cost saver. Over the years, market and technological forces pushed the 737 into ever-larger versions with increasing electronic and mechanical complexity. This is not, by any means, unique to the 737. Airliners constitute enormous capital investments both for the industries that make them and the customers who buy them, and they all go through a similar growth process.

Most of those market and technical forces are on the side of economics, not safety. They work as allies to relentlessly drive down what the industry calls "seat-mile costs"—the cost of flying a seat from one point to another.

Much had to do with the engines themselves. The principle of Carnot efficiency dictates that the larger and hotter you can make any heat engine, the more efficient it becomes. That's as true for jet engines as it is for chainsaw engines.

It's as simple as that. The most effective way to make an engine use less fuel per unit of power produced is to make it larger. That's why the Lycoming O-360 engine in my Cessna has pistons the size of dinner plates. That's why marine diesel engines stand three stories tall. And that's why Boeing wanted to put the huge CFM International LEAP engine in its latest version of the 737.

There was just one little problem: The original 737 had (by today's standards) tiny little engines, which easily cleared the ground beneath the wings. As the 737 grew and was fitted with bigger engines, the clearance between the engines and the ground started to get a little…um, tight.

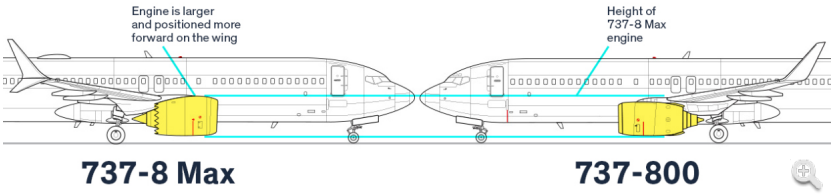

Illustration: Norebbo.com

By substituting a larger engine, Boeing changed the intrinsic aerodynamic nature of the 737 airliner.

Various hacks (as we would call them in the software industry) were developed. One of the most noticeable to the public was changing the shape of the engine intakes from circular to oval, the better to clear the ground.

With the 737 Max, the situation became critical. The engines on the original 737 had a fan diameter (that of the intake blades on the engine) of just 100 centimeters (40 inches); those planned for the 737 Max have 176 cm. That's a centerline difference of well over 30 cm (a foot), and you couldn't "ovalize" the intake enough to hang the new engines beneath the wing without scraping the ground.

The solution was to extend the engine up and well in front of the wing. However, doing so also meant that the centerline of the engine's thrust changed. Now, when the pilots applied power to the engine, the aircraft would have a significant propensity to "pitch up," or raise its nose.

The angle of attack is the angle between the wings and the airflow over the wings. Think of sticking your hand out of a car window on the highway. If your hand is level, you have a low angle of attack; if your hand is pitched up, you have a high angle of attack. When the angle of attack is great enough, the wing enters what's called an aerodynamic stall. You can feel the same thing with your hand out the window: As you rotate your hand, your arm wants to move up like a wing more and more until you stall your hand, at which point your arm wants to flop down on the car door.

This propensity to pitch up with power application thereby increased the risk that the airplane could stall when the pilots "punched it" (as my son likes to say). It's particularly likely to happen if the airplane is flying slowly.

Worse still, because the engine nacelles were so far in front of the wing and so large, a power increase will cause them to actually produce lift, particularly at high angles of attack. So the nacelles make a bad problem worse.

I'll say it again: In the 737 Max, the engine nacelles themselves can, at high angles of attack, work as a wing and produce lift. And the lift they produce is well ahead of the wing's center of lift, meaning the nacelles will cause the 737 Max at a high angle of attack to go to a *higher* angle of attack. This is aerodynamic malpractice of the worst kind.

Pitch changes with power changes are common in aircraft. Even my little Cessna pitches up a bit when power is applied. Pilots train for this problem and are used to it. Nevertheless, there are limits to what safety regulators will allow and to what pilots will put up with.

Pitch changes with increasing angle of attack, however, are quite another thing. An airplane approaching an aerodynamic stall cannot, under any circumstances, have a tendency to go further into the stall. This is called "dynamic instability," and the only airplanes that exhibit that characteristic—fighter jets—are also fitted with ejection seats.

Everyone in the aviation community wants an airplane that flies as simply and as naturally as possible. That means that conditions should not change markedly, there should be no significant roll, no significant pitch change, no nothing when the pilot is adding power, lowering the flaps, or extending the landing gear.

The airframe, the hardware, should get it right the first time and not need a lot of added bells and whistles to fly predictably. This has been an aviation canon from the day the Wright brothers first flew at Kitty Hawk.

Apparently the 737 Max pitched up a bit too much for comfort on power application as well as at already-high angles of attack. It violated that most ancient of aviation canons and probably violated the certification criteria of the U.S. Federal Aviation Administration. But instead of going back to the drawing board and getting the airframe hardware right (more on that below), Boeing relied on something called the "Maneuvering Characteristics Augmentation System," or MCAS.

Boeing's solution to its hardware problem was software.

I will leave a discussion of the corporatization of the aviation lexicon for another article, but let's just say another term might be the "Cheap way to prevent a stall when the pilots punch it," or CWTPASWTPPI, system. Hmm. Perhaps MCAS is better, after all.

MCAS is certainly much less expensive than extensively modifying the airframe to accommodate the larger engines. Such an airframe modification would have meant things like longer landing gear (which might not then fit in the fuselage when retracted), more wing dihedral (upward bend), and so forth. All of those hardware changes would be horribly expensive.

"Everything about the design and manufacture of the Max was done to preserve the myth that 'it's just a 737.' Recertifying it as

a new aircraft would have taken years and millions of dollars. In fact, the pilot licensed to fly the 737 in 1967 is still licensed to fly all subsequent versions of the 737."

—Feedback on an earlier draft of this article from a 737 pilot for a major airline

What's worse, those changes could be extensive enough to require not only that the FAA recertify the 737 but that Boeing build an entirely new aircraft. Now we're talking *real* money, both for the manufacturer as well as the manufacturer's customers.

That's because *the* major selling point of the 737 Max is that it is just a 737, and any pilot who has flown other 737s can fly a 737 Max without expensive training, without recertification, without another type of rating. Airlines—Southwest is a prominent example—tend to go for one "standard" airplane. They want to have one airplane that all their pilots can fly because that makes both pilots and airplanes fungible, maximizing flexibility and minimizing costs.

It all comes down to money, and in this case, MCAS was the way for both Boeing and its customers to keep the money flowing in the right direction. The necessity to insist that the 737 Max was no different in flying characteristics, no different in systems, from any other 737 was the key to the 737 Max's fleet fungibility. That's probably also the reason why the documentation about the MCAS system was kept on the down-low.

Put in a change with too much visibility, particularly a change to the aircraft's operating handbook or to pilot training, and someone—probably a pilot—would have piped up and said, "Hey. This doesn't look like a 737 anymore." And then the money would flow the wrong way.

As I explained, you can do your own angle-of-attack experiments just by putting your hand out a car door window and rotating it. It turns out that sophisticated aircraft have what is essentially the mechanical equivalent of a hand out the window: the angle-of-attack sensor.

You may have noticed this sensor when boarding a plane. There are usually two of them, one on either side of the plane, and usually just below the pilot's windows. Don't confuse them with the pitot tubes (we'll get to those later). The angle-of-attack sensors look like wind vanes, whereas the pitot tubes look like, well, tubes.

Angle-of-attack sensors look like wind vanes because that's exactly what they are. They are mechanical hands designed to rotate in response to changes in that angle of attack.

The pitot tubes measure how much the air is "pressing" against the airplane, whereas the angle-of-attack sensors measure what direction that air is coming from. Because they measure air pressure, the pitot tubes are used to determine the aircraft's speed through the air. The angle-of-attack sensors measure the aircraft's direction relative to that air.

There are two sets of angle-of-attack sensors and two sets of pitot tubes, one set on either side of the fuselage. Normal usage is to have the set on the pilot's side feed the instruments on the pilot's side and the set on the copilot's side feed the instruments on the copilot's side. That gives a state of natural redundancy in instrumentation that can be easily cross-checked by either pilot. If the copilot thinks his airspeed indicator is acting up, he can look over to the pilot's airspeed indicator and see if it agrees. If not, both pilot and copilot engage in a bit of triage to determine which instrument is profane and which is sacred.

Long ago there was a joke that in the future planes would fly themselves, and the only thing in the cockpit would be a pilot and a dog. The pilot's job was to make the passengers comfortable that someone was up front. The dog's job was to bite the pilot if he tried to touch anything.

On the 737, Boeing not only included the requisite redundancy in instrumentation and sensors, it also included redundant flight computers—one on the pilot's side, the other on the copilot's side. The flight computers do a lot of things, but their main job is to fly the plane when commanded to do so and to make sure the human pilots don't do anything wrong when they're flying it. The latter is called "envelope protection."

Let's just call it what it is: the bitey dog.

Let's review what the MCAS does: It pushes the nose of the plane down when the system thinks the plane might exceed its angle-of-attack limits; it does so to avoid an aerodynamic stall. Boeing put MCAS into the 737 Max because the larger engines and their placement make a stall more likely in a 737 Max than in previous 737 models.

When MCAS senses that the angle of attack is too high, it commands the aircraft's trim system (the system that makes the plane go up or down) to lower the nose. It also does something else: Indirectly, via something Boeing calls the "Elevator Feel Computer," it pushes the pilot's control columns (the things the pilots pull or push on to raise or lower the aircraft's nose) downward.

In the 737 Max, like most modern airliners and most modern cars, everything is monitored by computer, if not directly controlled by computer. In many cases, there are no actual mechanical connections (cables, push tubes, hydraulic lines) between the pilot's controls and the things on the wings, rudder, and so forth that actually make the plane move. And, even where there are mechanical connections, it's up to the computer to determine if the pilots are engaged in good decision making (that's the bitey dog again).

But it's also important that the pilots get physical feedback about what is going on. In the old days, when cables connected the pilot's controls to the flying surfaces, you had to pull up, hard, if the airplane was trimmed to descend. You had to push, hard, if the airplane was trimmed to ascend. With computer oversight there is a loss of natural sense in the controls. In the 737 Max, there is no real "natural feel."

True, the 737 does employ redundant hydraulic systems, and those systems do link the pilot's movement of the controls to the action of the ailerons and other parts of the airplane. But those hydraulic systems are powerful, and they do not give the pilot direct feedback from the aerodynamic forces that are acting on the ailerons. There is only an artificial feel, a feeling that the computer wants the pilots to feel. And sometimes, it doesn't feel so great.

When the flight computer trims the airplane to descend, because the MCAS system thinks it's about to stall, a set of motors and jacks push the pilot's control columns forward. It turns out that the Elevator Feel Computer can put a *lot* of force into that column—indeed, so much force that a human pilot can quickly become exhausted trying to pull the column back, trying to tell the computer that this really, really should not be happening.

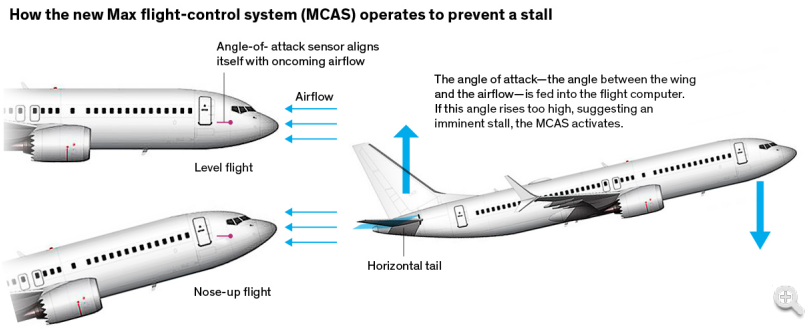

Illustration: Norebbo.com

The antistall system depended crucially on sensors that are installed on each side of the airliner—but the system consulted only the sensor on one side.

Indeed, not letting the pilot regain control by pulling back on the column was an explicit design decision. Because if the pilots could pull up the nose when MCAS said it

should go down, why have MCAS at all?

MCAS is implemented in the flight management computer, even at times when the autopilot is turned off, when the pilots think they are flying the plane. In a fight between the flight management computer and human pilots over who is in charge, the computer will bite humans until they give up and (literally) die.

Finally, there's the need to keep the very existence of the MCAS system on the hush-hush lest someone say, "Hey, this isn't your father's 737," and bank accounts start to suffer.

The flight management computer is a computer. What that means is that it's not full of aluminum bits, cables, fuel lines, or all the other accoutrements of aviation. It's full of lines of code. And that's where things get dangerous.

Those lines of code were no doubt created by people at the direction of managers. Neither such coders nor their managers are as in touch with the particular culture and mores of the aviation world as much as the people who are down on the factory floor, riveting wings on, designing control yokes, and fitting landing gears. Those people have decades of institutional memory about what has worked in the past and what has not worked. Software people do not.

In the 737 Max, only one of the flight management computers is active at a time—either the pilot's computer or the copilot's computer. And the active computer takes inputs *only* from the sensors on its own side of the aircraft.

When the two computers disagree, the solution for the humans in the cockpit is to look across the control panel to see what the other instruments are saying and then sort it out. In the Boeing system, the flight management computer does not "look across" at the other instruments. It believes only the instruments on its side. It doesn't go old-school. It's modern. It's software.

This means that if a particular angle-of-attack sensor goes haywire—which happens all the time in a machine that alternates from one extreme environment to another, vibrating and shaking all the way—the flight management computer just believes it.

It gets even worse. There are several other instruments that can be used to determine things like angle of attack, either directly or indirectly, such as the pitot tubes, the artificial horizons, etc. All of these things would be cross-checked by a human pilot to quickly diagnose a faulty angle-of-attack sensor.

In a pinch, a human pilot could just look out the windshield to confirm visually and directly that, no, the aircraft is not pitched up dangerously. That's the ultimate check and should go directly to the pilot's ultimate sovereignty. Unfortunately, the current implementation of MCAS denies that sovereignty. It denies the pilots the ability to respond to what's before their own eyes.

Like someone with narcissistic personality disorder, MCAS gaslights the pilots. And it turns out badly for everyone. "Raise the nose, HAL." "I'm sorry, Dave, I'm afraid I can't do that."

In the MCAS system, the flight management computer is blind to any other evidence that it is wrong, including what the pilot sees with his own eyes and what he does when he desperately tries to pull back on the robotic control columns that are biting him, and his passengers, to death.

In the old days, the FAA had armies of aviation engineers in its employ. Those FAA employees worked side by side with the airplane manufacturers to determine that an airplane was safe and could be certified as airworthy.

As airplanes became more complex and the gulf between what the FAA could pay and what an aircraft manufacturer could pay grew larger, more and more of those engineers migrated from the public to the private sector. Soon the FAA had no in-house ability to determine if a particular airplane's design and manufacture were safe. So the FAA said to the airplane manufacturers, "Why don't you just have your people tell us if your designs are safe?"

The airplane manufacturers said, "Sounds good to us." The FAA said, "And say hi to Joe, we miss him."

Thus was born the concept of the "Designated Engineering Representative," or DER. DERs are people in the employ of the airplane manufacturers, the engine manufacturers, and the software developers who certify to the FAA that it's all good.

Now this is not quite as sinister a conflict of interest as it sounds. It is in nobody's interest that airplanes crash. The industry absolutely relies on the public trust, and every crash is an existential threat to the industry. No manufacturer is going to employ DERs that just pencil-whip the paperwork. On the other hand, though, after a long day and after the assurance of some software folks, they might just take their word that things will be okay.

It is astounding that no one who wrote the MCAS software for the 737 Max seems even to have raised the possibility of using multiple inputs, including the opposite angle-of-attack sensor, in the computer's determination of an impending stall. As a lifetime member of the software development fraternity, I don't know what toxic combination of inexperience, hubris, or lack of cultural understanding led to this mistake.

But I do know that it's indicative of a much deeper problem. The people who wrote the code for the original MCAS system were obviously terribly far out of their league and did not know it. How can they implement a software fix, much less give us any comfort that the rest of the flight management software is reliable?

So Boeing produced a dynamically unstable airframe, the 737 Max. That is big strike No. 1. Boeing then tried to mask the 737's dynamic instability with a software system. Big strike No. 2. Finally, the software relied on systems known for their propensity to fail (angle-of-attack indicators) and did not appear to include even rudimentary provisions to cross-check the outputs of the angle-of-attack sensor against other sensors, or even the other angle-of-attack sensor. Big strike No. 3.

None of the above should have passed muster. None of the above should have passed the "OK" pencil of the most junior engineering staff, much less a DER.

That's not a big strike. That's a political, social, economic, and technical sin.

It just so happens that, during the timeframe between the first 737 Max crash and the most recent 737 crash, I'd had the occasion to upgrade and install a brand-new digital autopilot in my own aircraft. I own a 1979 Cessna 172, the most common aircraft in history, at least by production numbers. Its original certification also predates that of the 737's by about a decade (1955 versus 1967).

My new autopilot consists of several very modern components, including redundant flight computers (dual Garmin G5s) and a sophisticated communication "bus" (a Controller Area Network bus) that lets all the various components talk to one another, irrespective of where they are located in my plane. A CAN bus derives from automotive "drive by wire" technology but is otherwise very similar in purpose and form to the various ARINC buses that connect the components in the 737 Max.

My autopilot also includes electric pitch trim. That means it can make the same types of configuration changes to my 172 that the flight computers and MCAS system make to the 737 Max. During the installation, after the first 737 Max crash, I remember remarking to a friend that it was not lost on me that I was potentially adding a hazard similar to the one that brought down the Lion Air crash.

Finally, my new autopilot also implements "envelope protection," the envelope being the graph of the performance limitations of an aircraft. If my Cessna is *not* being flown by the autopilot, the system nonetheless constantly monitors the airplane to make sure that I am not about to stall it, roll it inverted, or a whole host of other things. Yes, it has its own "bitey dog" mode.

As you can see, the similarities between my US $20,000 autopilot and the multimillion-dollar autopilot in every 737 are direct, tangible, and relevant. What, then, are the differences?

For starters, the installation of my autopilot required paperwork in the form of a "Supplemental Type Certificate," or STC. It means that the autopilot manufacturer and the FAA both agreed that my 1979 Cessna 172 with its (Garmin) autopilot was so significantly different from what the airplane was when it rolled off the assembly line that it was *no longer the same Cessna 172*. It was a different aircraft altogether.

In addition to now carrying a new (supplemental) aircraft-type certificate (and certification), my 172 required a very large amount of new paperwork to be carried in the plane, in the form of revisions and addenda to the aircraft operating manual. As you can guess, most of those addenda revolved around the autopilot system.

Of particular note in that documentation, which must be studied and understood by anyone who flies the plane, are various explanations of the autopilot system, including its command of the trim control system and its envelope protections.

There are instructions on how to detect when the system malfunctions *and how to disable the system, immediately*. Disabling the system means pulling the autopilot circuit breaker; instructions on how to do that are strewn throughout the documentation, repeatedly. Every pilot who flies my plane becomes intimately aware that it is *not* the same as any other 172.

This is a big difference between what pilots who want to fly my plane are told and what pilots stepping into a 737 Max are (or were) told.

Another difference is between the autopilots in my system and that in the 737 Max. All of the CAN bus–interconnected components constantly do the kind of instrument cross-check that human pilots do and that, apparently, the MCAS system in the 737 Max does not. For example, the autopilot itself has a self-contained attitude platform that checks the attitude information coming from the G5 flight computers. If there is a disagreement, the system simply goes off-line and alerts the pilot that she is now flying manually. It doesn't point the airplane's nose at the ground, thinking it's about to stall.

Perhaps the biggest difference is in the amount of physical force it takes for the pilot to override the computers in the two planes. In my 172, there are still cables linking the controls to the flying surfaces. The computer has to press on the same things that I have to press on—and its strength is nowhere near as great as mine. So even if, say, the computer thought that my plane was about to stall when it wasn't, I can easily overcome the computer.

In my Cessna, humans still win a battle of the wills every time. That used to be a design philosophy of every Boeing aircraft, as well, and one they used against their archrival Airbus, which had a different philosophy. But it seems that with the 737 Max, Boeing has changed philosophies about human/machine interaction as quietly as they've changed their aircraft operating manuals.

The 737 Max saga teaches us not only about the limits of technology and the risks of complexity, it teaches us about our real priorities. Today, safety doesn't come first—money comes first, and safety's only utility in that regard is in helping to keep the money coming. The problem is getting worse because our devices are increasingly dominated by something that's all too easy to manipulate: software.

Hardware defects, whether they are engines placed in the wrong place on a plane or O-rings that turn brittle when cold, are notoriously hard to fix. And by hard, I mean expensive. Software defects, on the other hand, are easy and cheap to fix. All you need to do is post an update and push out a patch. What's more, we've trained consumers to consider this normal, whether it's an update to my desktop operating systems or the patches that get posted automatically to my Tesla while I sleep.

Back in the 1990s, I wrote an article comparing the relative complexity of the Pentium processors of that era, expressed as the number of transistors on the chip, to the complexity of the Windows operating system, expressed as the number of lines of code. I found that the complexity of the Pentium processors and the contemporaneous Windows operating system was roughly equal.

That was the time when early Pentiums were affected by what was known as the FDIV bug. It affected only a tiny fraction of Pentium users. Windows was also affected by similar defects, also affecting only fractions of its users.

But the effects on the companies were quite different. Where Windows addressed its small defects with periodic software updates, in 1994 Intel recalled the (slightly) defective processors. It cost the company $475 million—more than $800 million in today's money.

I believe the relative ease—not to mention the lack of tangible cost—of software updates has created a cultural laziness within the software engineering community. Moreover, because more and more of the hardware that we create is monitored and controlled by software, that cultural laziness is now creeping into hardware engineering—like building airliners. Less thought is now given to getting a design correct and simple up front because it's so easy to fix what you didn't get right later.

Every time a software update gets pushed to my Tesla, to the Garmin flight computers in my Cessna, to my Nest thermostat, and to the TVs in my house, I'm reminded that none of those things were complete when they left the factory—because their builders realized they didn't have to be complete. The job could be done at any time in the future with a software update.

"I'm a software developer turned network engineer and have written airliner avionics software in the past. It was interesting how many hoops we had to jump through to get an add-on board for the computer certified, while software certifications were nil (other than "cannot run on Windows," "must be written in C++"). This was, admittedly, nearly 10 years ago, and I hope that things have changed since."

—Anonymous, personal correspondence

Boeing is in the process of rolling out a set of software updates to the 737 Max flight control system, including MCAS. I don't know, but I suspect that those updates will center on two things:

1. Having the software "cross-check" system indicators, just as a human pilot would. Meaning, if one angle-of-attack indicator says the plane's about to stall, but the other one says it's not so, at least hold off judgment about pushing the nose down into the dirt and maybe let a pilot or two know you're getting conflicting signals.
2. Backing off on the "shoot first, ask questions later" design philosophy—meaning, looking at multiple inputs.

For the life of me, I do not know why those two basic aviation design considerations, bedrocks of a mind-set that has served the industry so well until now, were not part of the original MCAS design. And, when they were not, I do not know or understand what part of the DER process failed to catch the fundamental design defect.

But I suspect that it all has to do with the same thing that brought us from Boeing's initial desire to put larger engines on the 737 and to avoid having to internalize the cost of those larger engines—in other words, to do what every child is taught is impossible: get a free lunch.

The emphasis on simplicity comes from the work of Charles Perrow, a sociologist at Yale University whose 1984 book, *Normal Accidents: Living With High-Risk Technologies*, tells it all in the very title. Perrow argues that system failure is a normal outcome in any system that is very complex and whose components are "tightly bound"—meaning that the behavior of one component immediately controls the behavior of another. Though such failures may seem to stem from one or another faulty part or practice, they must be seen as inherent in the system itself. They are "normal" failures.

Nowhere is this problem more acutely felt than in systems designed to augment or improve safety. Every increment, every increase in complexity, ultimately leads to decreasing rates of return and, finally, to negative returns. Trying to patch and then repatch such a system in an attempt to make it safer can end up making it less safe.

This is the root of the old engineering axiom "Keep it simple, stupid" (KISS) and its aviation-specific counterpart: "Simplify, then add lightness."

The original FAA Eisenhower-era certification requirement was a testament to simplicity: Planes should not exhibit significant pitch changes with changes in engine power. That requirement was written when there was a direct connection between the controls in the pilot's hands and the flying surfaces on the airplane. Because of that, the requirement—when written—rightly imposed a discipline of simplicity on the design of the airframe itself. Now software stands between man and machine, and no one seems to know exactly what is going on. Things have become too complex to understand.

I cannot get the parallels between the 737 Max and the space shuttle Challenger out of my head. The Challenger accident, another textbook case study in normal failure, came about not because people didn't follow the rules but because they did. In the Challenger case, the rules said that they had to have prelaunch conferences to ascertain flight readiness. It didn't say that a significant input to those conferences couldn't be the political considerations of delaying a launch. The inputs were weighed, the process was followed, and a majority consensus was to launch. And seven people died.

In the 737 Max case, the rules were also followed. The rules said you couldn't have a large pitch-up on power change and that an employee of the manufacturer, a DER, could sign off on whatever you came up with to prevent a pitch change on power change. The rules didn't say that the DER couldn't take the business considerations into the decision-making process. And 346 people are dead.

It is likely that MCAS, originally added in the spirit of increasing safety, has now killed more people than it could have ever saved. It doesn't need to be "fixed" with more complexity, more software. It needs to be removed altogether.

An earlier version of this article was cited in *EE Times*.

Editor's note: This story was updated on 21 April to clarify that the MCAS pushes the airliner's nose by means of the "Elevator Feel Computer."

About the Author

Gregory Travis is a writer, a software executive, a pilot, and an aircraft owner. In 1977, at the age of 13, he wrote Note, one of the first social media platforms, and he has logged more than 2,000 hours of flying time, ranging from gliders to a Boeing 757 (as a full-motion simulator).