



“Clear to land” should be followed by a glance out the window to see if the runway is actually “clear”; in other words, *trust but verify*

Don't Believe Everything You See or Hear

Don't Believe Everything You See or Hear¹

Tom Love

ShouldersCorp

“Recent high-profile accidents have demonstrated that even when redundant or back-up systems are present, they don't guarantee a successful outcome.”

R. Scott Puddy
AvWeb Article, January 10, 2001

¹ This is a chapter from Tom Love's forthcoming book, Software Pilots. Please do not redistribute without the author's permission.

Don't Believe Everything You See or Hear

As a driver of a modern day automobile, we are accustomed to sitting in front of an apparently simple, yet highly automated instrument panel. This simple instrument panel may go completely unmonitored for short trips unless a warning light begins to blink or a bell alerts us to a problem. On longer trips gauges indicating speed, remaining fuel, and inside or outside temperature are likely to be monitored at some frequency. If we see a patrol car with radar ahead, we are very likely to monitor the speedometer quite closely for a few minutes.

To make the car go forward, we should apply the brake, crank the car, shift the transmission into a forward gear, release the brake, press the accelerator and steer as required. As the car begins to move, we expect to see the tachometer increase, the speedometer increase from zero, and the odometer begin to rollover after we have traveled a tenth of a mile or so.

In modern day automobiles, we have grown to trust our gauges or to be more precise, the software agents monitoring critical systems. Some of us even trust the digital readout which says "2 miles to empty".

But airplanes and software projects don't operate so predictably!

One of the more complicated aspects of learning to fly an airplane is to learn to systematically monitor a lot of dials, gauges and instrumentation in the airplane and to learn that these "gauges" can be surprisingly **inaccurate!** The vertical speed indicator lags a change in pitch by about 9 seconds. The magnetic compass can be off by 40 degrees. The directional gyro (provided to solve the errors associated with the magnetic compass) can be arbitrarily wrong depending upon the heading set by the pilot. The non-directional beacon indicator could be pointing to a thunderstorm instead of to an airport where you wanted to land to wait out the storm. This is only a partial list of gauges that can be inaccurate and this list assumes that everything in the airplane is working properly. The challenge for the pilot increases substantially when already inaccurate instruments start failing individually or collectively due to electrical or mechanical failures or both!

Only recently have general aviation airplanes begun to add automated warnings for primary instruments or systems, like low fuel, low voltage, or defective vacuum pumps. More sophisticated warnings based upon more complex measurements are being added to 21st Century airplanes as flat LCD displays and computers replace a cockpit full of dials and gauges.

In a similar way, software project managers are flooded with raw information -- most of which is inaccurate and often irrelevant. Most software managers have no real time instrument panel and nothing resembling an automated warning system. Instead they rely upon their programmers to tell them that, "I'm 90% done". Other examples of highly inaccurate information include the following:

- "Not to worry, I understand how to do object-oriented design"

Don't Believe Everything You See or Hear

- “We don't need to write any specs, because we understand exactly what the customer needs”
- “It'll all be working Friday”
- “Sure, I tested it all myself”
- “I found a really nice way to solve that problem we were talking about on Friday and it only required 35 new classes which I coded-up over the weekend”
- “Of course, I documented my code, I have one comment in each class”
- “I admit the schedule is a little aggressive, but I'm sure we'll be able to test 650 classes in 17 days”
- And another classic, “we don't have to worry about that”.

Of course the software manager has to look outside the cockpit as well. The outside is filled with “stakeholders” – company executives, customers, marketing and sales staff, financial managers, investors, and “the competition”. So the software manager finds himself in a high-speed dogfight with high performance devices all around them and one slow moving gauge reporting on the budget. This one gauge usually lags real time by a week or more.

A software pilot needs a real time project panel with the specialized gauges required for a particular type of project. The software pilot also needs the most reliable and timely information displayed on those gauges; he needs to know how to scan and interpret the gauges; and the software pilot needs to know how to detect erroneous information or defective gauges and what to do about such errors.

In this chapter we'll describe some of the problems that exist with the gauges on the panel of an airplane. The purpose is to provide analogies applicable to the task of “landing a software project”. Then we'll design a “panel” for a software project and discuss how a software pilot should use such a panel.

Flying with Inaccurate Information

So how do you fly an airplane with such inaccurate information? It's not easy. Much of the advanced training that a pilot receives (beyond a private pilot certificate) is actually accommodating for inaccurate information on cockpit instruments.

Let's consider a couple of very basic instruments – the altimeter and the directional gyro. The purpose of the altimeter is to measure altitude above sea level. Note that what you are really interested in is how far above the ground or ground based obstacles you are. Altimeters do not provide this information. The pilot has to consult maps to determine where he thinks he is and how far

Don't Believe Everything You See or Hear

above an obstacle he might be. Prior to GPSs, there was no accurate knowledge of where the plane was above the ground, let alone its exact position in 3-D space².

The directional gyro (DG) was introduced into airplanes to correct the horrendous inaccuracy of magnetic compasses. On a recent training flight I observed a magnetic compass being off by 60 degrees in a steep descending turn. The DG provides a high-speed gyroscope and an instrument to detect changes from a given orientation. The problem is that DGs accumulate errors as you are flying along, unless a more complicated instrument is added to correct these errors. So the new pilot has to remember to reset the DG every 15 minutes or so when the airplane is in straight and level flight. In this way the DG should never be off by more than a few degrees, assuming everything is working correctly.

The altimeter has the problem that it is really only a barometer with its pressure reading expressed in altitude – feet above sea level. To achieve the desired accuracy, the pilot must continuously monitor air traffic control or automatic weather reporting stations to find the current barometric pressure and adjust the altimeter accordingly. Swings in barometric pressure can result in 100's of feet of error in the altimeter. Since planes sometimes fly with only 500 feet of vertical separation, this is very important. It's critically important when flying in the clouds with little if any visibility.

Of course, there are some important special cases. If flying above 18,000 feet in the United States, the altimeter must be set to 29.92 – so called standard pressure.

“So are you telling me that a bunch of general aviation planes are up there flying around not knowing very accurately where they are and which direction they are going.” That is correct.

The pilot of a typical, mid-1980's general aviation airplane flying in VFR conditions has to rely upon rudimentary DGs and altimeters for primary navigation. In other words, most were off course most of the time!

Radio navigation devices became widespread in the 1940's. They were better than DGs and altimeters, but their potential errors were huge. Basic radio beacons were used for homing – a great advance at the time. But it was discovered that these devices were notoriously inaccurate.

In the 1950's, Very-high frequency Omnidirectional Radios (VORs) were introduced. The VOR transmitted a signal for each radial and the pilot could “dial

² Loran and various other navigation tools were designed and implemented to improve a pilot's spatial awareness, but they are not very accurate. WAAS compatible GPSs are accurate to 3 meters in X, Y, and Z coordinates. A Loran is accurate to a mile or so, which I do not consider very accurate. Instrument landing systems can be very accurate but are only available within a few miles of some airports. Some high performance planes will have a radar altimeter which actually measures the height above terrain.

Don't Believe Everything You See or Hear

in a radial” and track it to the station. This system works well and is in continuous use for commercial air traffic today. But professional pilots know that VORs are very accurate 10 miles from the station, yet incredibly inaccurate within a mile or two of the station. Since it is possible for VORs transmitting on the same frequency to be picked up by a single airplane, the system relies upon “morse code identifiers” to make sure you are tracking to or from the right VOR.

A few days ago, I boarded a commercial airliner (Boeing 737-300) with 175 of “my closest friends”. The plane was designed in the 1970’s, manufactured in the 1980’s and flown by a pilot trained in the 1990’s. As I boarded the plane, I glanced into the cockpit. It’s chock a bloc with instruments. I now know that most are highly inaccurate. The on board GPS (which is not WAAS compatible) is the most reliable instrument on board. Yet, it is not very reliable in it’s determination of altitude. It’s X and Y coordinates are accurate to about 10 meters; it’s Z coordinate accurate to 50 meters.

One of the reasons that commercial airliners have so many gauges and displays is that no one is very accurate. So the pilot needs a lot of information from a variety of sources to cross check accuracy. Of course, the more instruments the higher the probability that one instrument will fail on any given flight. So we provide a large number of inaccurate and likely to fail instruments to “help the pilot”. It’s sounding more and more like a software project!

By contrast, the approach taken by technically advanced airplane manufacturers (like Cirrus, Lancair, Eclipse, Adam, etc.) is to provide many fewer instruments of much higher reliability and accuracy, which are replicated as required to achieve the specified level of reliability. This more accurate information can be used to guide the pilot far more accurately. The pilot’s job becomes keeping the little plane in a sequence of boxes displayed in 3-D on the large color screen in front of him or her. This “highway in the sky system” is being evaluated for use in general aviation airplanes as this is being written. Some version of it will be in widespread use in a few years.

So what are the big lessons from the aviation world that we can use in the software business? If you have inaccurate information, getting a lot more of it is not helpful. What you need is accurate information which is timely and which is succinctly presented so that the right actions can be taken at each “juncture” of the project. It would also be helpful if Software Pilots were trained to provide proper control inputs based upon observed patterns of sometimes imperfect information on their project panel.

That is so easy to say. How on earth can we do it on a real project?

Managing a Software Project with Inaccurate Information

Software managers are quite accustomed to complaining about the lack of metrics on the projects they manager. “It’s their own fault!” It’s right in front of

Don't Believe Everything You See or Hear

them. They just need to learn where to look and what they are seeing. In other words, a lot of “pilot training” is required coupled with real time instruments.

Software projects involve a collection of human beings spending most of their days in front of a display reading information and occasionally entering information via the keyboard. This information is often time-stamped (to a millisecond of accuracy) and stored on a shared disk drive available to everyone on the project. As a result, vast amounts of “raw data” are available. Very little of it is ever used. That's too bad.

On everyone's computer these days, there are tools that can be used to count various things based upon this “raw data”. Any tools that aren't already on the computer can be quickly downloaded from the Internet. How many classes have been designed? How many lines of documentation exist for each class? How many test cases have been written? How many classes have been exhaustively tested? How many customer error reports have been submitted? How long does it take us to “fix” a typical customer error report? How many lines of code have been “written” so far on the project?

Alarms go off for many software managers when any suggestion is made that we “count lines of code”. My good friend, Capers Jones, thinks project managers should have their license revoked if they count lines of code. I disagree. I think they should have their license “pulled” if they aren't continuously counting something! I want to know that we only have an average of 10 lines of code per class in the configuration management systems this morning. I want to know that the average class is 3400 lines of code. I want to know if we have an average of 1200 methods per class.³

Actually what Capers objects to is estimating effort for a software project based upon number of lines of code, since it has been shown to be an unreliable predictor, used in isolation. I fully understand the basis for his objection; my problem is that all the proposed solutions are less accurate. I think we can use objective measures of “bulk”, like lines of code, as long as we do so in a knowledgeable way.

The stable measure of work in a modern software application turns out to be the “class”. It so happens that classes tend to have about the same number of lines of code, tend to take about the same effort to code and test, and tend to have a consistent number of methods per class across object-oriented programming languages, if properly used.

So what should a software pilot's panel look like?

A Design for a Software Pilot's “Panel”

There is not one design for a Software Pilot's Panel. The ideal panel for a recreational airplane seating two people is radically different from the panel for a

³ See Object Lessons, Chapter 14, for the expected metrics for an object-oriented program.

Don't Believe Everything You See or Hear

corporate jet carrying a dozen people across the country at 325 knots or one for a SR-72 Blackbird reconnaissance aircraft flying at 80,000 feet. Panels in aircraft are designed to suit the mission requirements of the airplane. The panel for a complex airplane like the SR-72 must be complex. Sometimes the panels become so complex that they require a flight crew to monitor and control the mission.

So, there is no such thing as “the right panel” for a software project. Rather there are panels customized for the project at hand. The panel might even change during a project as the emphasis changes.

Let's begin with the design of a panel for a 10-person software project working to deliver 50 Java classes in 100 business days. We want gauges that display the following information on a project website:

- Cost for the project – expended, projected
- Days until delivery
- Number of Java Classes designed, coded, tested and documented – gathered in real time from the configuration management system
- Number of pages of documentation required versus number written and QA-ed
- Number of outstanding problem reports by severity
- Rate of servicing problem reports by severity
- Number of screens designed, coded and tested
- Number of database tables designed, coded and tested
- Number of reports designed, coded and tested
- Projected completion date based upon above information
- Critical assumptions: e.g., budget fixed, schedule fixed, resources flexible, etc.

Here is a prototype for such a Software Pilot's Panel:

[SW Pilots Panel.htm](#)

Interpreting Information Displayed on the Panel

Pilots spend weeks learning how to interpret the information displayed on each gauge, how to cross check the information displayed on different gauges, and how to scan the panel depending upon the stage of flight or the particular maneuver being attempted.

Software project managers have no such training and typically do not even have a “panel” displaying the most basic real time information for their project. Pilots also get special training in “partial panel” flight – what to do if one or more instruments fails in flight. Software managers are trying to land projects without any panel at all.

Don't Believe Everything You See or Hear

Do you suppose it's significant that the Wright flyer was the first airplane with an instrument panel? One of the critical breakthroughs for the Wright brothers was the importance of "aircraft control". Other problems related to lift and propulsion had been solved. But keeping it in the air amidst gusty winds and landing it predictably required control. Proper control required instruments to determine the magnitude of the next control input.



Figure 1: Instrument panel of Wright flyer, circa 1903, as depicted in Microsoft Flight Simulator.

So software pilots need a simple instrument panel to control their projects. It's madness to attempt a flight without basic information and sufficient control mechanisms. Note that the typically "simulator pilot" requires more information than Orville, as shown in the upper left corner.

The 1903 flyer had three instruments on board:

- An anemometer to measure air speed
- A stop watch to measure flight duration
- A tachometer to measure the speed of the engine.

Pilots understand that certain combinations of instrument readings suggest a particular "control input". On downwind for landing in a Cessna 172 properly configured, the airspeed indicator should read 90 knots indicated airspeed (corrected for wind speed). This suggests that when reaching the arrival end of the runway (referred to as being "abeam the numbers"), the power should be

Don't Believe Everything You See or Hear

reduced to 1500 RPM, ten degrees of flaps should be added, and the pitch should be reduced to achieve a 500 feet per minute decent rate.

Software pilots should be trained to recognize certain situations and know exactly what combination of inputs are required. Imagine the following situation:

1. You take over a ten person project 60 days prior to delivery
2. 60 Java classes have been coded
3. All functionality has been demonstrated
4. Performance is less than anticipated (6 second response time, instead of 3 second response time)
5. No user level documentation has been developed

A trained software pilot should immediately know, "I can land this project as long as I don't expand the functionality or loose any time". The control inputs that a software pilot should anticipate include the following:

1. Testing and performance optimization are key to a successful landing, so the testing and optimization plan must be carefully and quickly reviewed and modified as necessary
2. Scope creep has to be aggressively managed – "we're building an application with 60 classes and only 60 classes!"; carefully monitor class count
3. A documentation team must be formed immediately and set to work; there should be plenty of time to accomplish what is required, if you start immediately and if the functionality doesn't change.

By contrast, let's consider another example (which really happened).

1. You are asked to take over a 20-person project developing 650 Java classes.
2. The team has been at work for 6 months.
3. No one on the team has ever delivered a Java application before.
4. Only 250 classes have been coded.
5. There are 45 days until the committed delivery date.
6. The "plan" calls for 17 days to test 650 classes with a test team of 7.

The trained software pilot knows exactly what to do. Declare an emergency! According the instruments, you are in the clouds, in a graveyard spiral, accelerating and quickly passing through 2500 feet. How do you know that?

Let's start from the bottom. Testing a class requires writing one or more test cases for each method. A typical class in a commercial application has 25 methods. How long does it take to write and verify 25 test cases? It takes at least two hours per test case. So assuming an 8 hour work day and maximum efficiency, 650 classes will (at best) require 4063 person days to test the component parts, plus some additional time to test sub-assemblies and to verify overall system performance and correctness. To be done with in 17 days, more

Don't Believe Everything You See or Hear

than 239 testers must be at work tomorrow. Adding 229 people in a day is rarely possible and unlikely to be successful⁴.

If we were still testing on day 44, there would be no time for repairing errors found or for documenting any changes. We also know that 400 classes are yet to be coded. We should be able to measure how long it has taken this team to develop 250 classes, then simply extrapolate to estimate when 400 more will be available for testing. Plausible estimates would suggest that a class takes 5 person days of effort to design, code and unit test. In 400 person weeks of effort, the code can be ready for test!

Since the project team has 7 testers, it can have no more than 12 developers and one project manager. So a 12-person team can develop about 108 classes in 45 days. Then the classes need to be tested, integrated, and documented.

Given the inexperience of the team with the programming language, we can assume that it will take more than the usual amount of time to develop and to test such an application. If the team has never used an object-oriented programming language, for example, you might consider doubling the effort and extending the development time considerably.

We know that 6 months of data is available on the productivity of this particular team. It must be recovered and used. Experience suggests that such data is invaluable in refining estimates. "Don't tell me how fast you think you are coding, tell me how fast you have actually coded for the first 6 months of the project."

Never forget that productivity declines rapidly as project size grows. The ideal size for a project team is two. So a 20-person team will necessarily work much slower than a 10-person team. There is also a size above which adding more people will not affect productivity; it will only affect costs. This incompressibility of the schedule has to be taken into account and properly respected. For example, adding 50 people to this project with 45 days remaining is not the solution! It's roughly equivalent to accelerating while in a graveyard spiral.

What Happens When the Instruments Become Inconsistent? Or just fail?

Airplane panels are connected to sensors. Software panels gather information from other human beings and from "software sensors". Neither set of connections is foolproof.

A software pilot should make every effort to remove the human factor from their panel wiring. Software managers are legendary for just walking around and asking, "tell me what percentage complete you are". The answer is worthless. By contrast if the software pilot has a panel connected to objective sources of information which are fed into calculators based upon experience, some really

⁴ Some organizations have 5 testers per developer.

Don't Believe Everything You See or Hear

useful information is being gathered which will allow the software pilot to take corrective action which might actually help land the project.

But, even with objective measures coming into the panel, there are opportunities for failure. Software pilots have to scan their instruments constantly asking themselves, "Does this information seem internally consistent?" At the slightest hint of a problem, immediate diagnostic procedures are required. Questions that may arise include:

- "Why were no new classes developed yesterday?"
- "Why do we have fewer classes coded today, than yesterday?"
- "Why do we have 20% more classes to code this afternoon, compared to this morning?"
- "Why is the rate of testing declining each day?"
- "Are we really finished with design, if we have only identified 5 methods per class?"
- "Why is average response time increasing during testing?"

Summary

It would be great if we could write a 10-page paper on how to "pilot a software project". But there is more to know than that. A one semester class is not sufficient either.

It's takes years to learn how to fly an airplane carrying 100 people. It should take about the same amount of training, study, testing, and practical experience to pilot a 100-person software project.

In this chapter, we have made the case for a "project panel", explained some of the gauges and dials that need to be present and even suggested a control or two. Over the next few years, we should gain a lot more experience with "project panels" and should settle on a few generic layouts for projects of various sizes and complexities.

These project panels are really nothing more than real time "control charts" as advocated by W. Edwards Deming, the man credited with revitalizing Japanese industry. Many of the other practices advocated here are remarkably similar to Deming's "Fourteen Points".

With any luck it won't take us 100 years to create consistent panels, trained pilots, and standardized control inputs for known patterns of real time information gathered on software projects. With these break-thrus of instrumentation and control systems, software pilots will be able to land as reliably as airline pilots.

Postscript

Do you suppose it's worth noting that this chapter never mentioned Microsoft Project? Landing a project involves a little planning, a lot of real time information monitoring and the ability to make split second decisions as required based upon the patterns of information displayed on the "project panel". Project planning tools play a small but vital role. Many project managers get confused and believe that work breakdown structures, Gantt charts and critical paths are all you need to know about managing projects. They would be much better off if they began to think of proper training, project panels, rapid decision-making and system engineering.