All.Net Analyst Report and Newsletter

Welcome to our Analyst Report and Newsletter

Iran(t) on merchantability for software

Actually, I rant on merchantability for software, but the search engines like Iran these days...

Software, so far, has somehow been ignored in terms of implied warranties of merchantability and fitness for purpose. The implied warranties state, essentially, that anything sold must be suitable for the ordinary purpose it is used for. [1] These cannot be waived for commercial goods, even if the contract asserts that they are. However, the standard is not perfection. For example, if the rest of the industry has some level of imperfection, then fitness is associated with that reality.

How good does that really have to be?

A lot of folks seem to be complaining today about the imperfection of software in withstanding unlimited assault from over the Internet and the supply chain. But I want to point something out. Where else do we have the expectation of perfection under these circumstances? I will focus on control systems for now, because they have obvious and direct physical effects and are more readily relatable.

Any control system component you can ever buy has its limits. Some environmental limits are generally specified (e.g., temperature, humidity, etc.) on the box or in the literature. If you run a system in some wrong environment, it will always fail. Put the compressor for the gas station completely under water and you are not likely to get any compressed air out. The machine will likely short circuit and break.

Not an apologist – an environmentalist

I am not an apologist for poor quality software. I think it is right to have implied warranties of merchantability apply to software. But I also think that nothing operates everywhere all the time. Software operated in an environment and if you put it in an unsuitable environment, it will likely fail just like anything else.

It is generally the job of the user/buyer to put things into suitable environments for their operation. If you put that pump underwater, it will fail, and if you put that SCADA [2] system on the Internet it will fail. That's because most control system components are designed to work in closed environmentally controlled environments. Both hardware and the software will break if you shoot at them all the time (hardware with missiles, software with packets).

You have to architect your information environment like you do your physical one if you want your software to work properly. Temperature controlled, humidity controlled, packet controlled... Send the right bit sequences and the software will work. Instruct it to blow itself up and it will.

The unanswered questions are “What software is suited to what environments?”, and “What should we reasonably expect of software?”

---

1 Uniform Commercial Code Article 2 – Sales (2002) Part 3 – General obligation and construction of contract § 2-314. Implied Warranty: Merchantability; Usage of Trade. (https://www.law.cornell.edu/ucc/2/2-314)

2 Supervisory Control and Data Acquisition (SCADA) systems are used to control many industrial processes.

What's in there?

As a first step in understanding modern software, it is important to understand that it is built up through a massive and largely uncontrolled supply chain. If you ask almost any supplier of software today to provide you with a list of all the components and their provenance (where they came from and how they were made recursively back to origins), they almost certainly cannot do it.

There are many reasons for this limitation. One of them is simply cost of goods sold. For example, I run some Web sites. They run on different hardware platforms, some that I own and control, and others that are outsourced to cloud providers. They run various version of Linux, which is composed of software from at least tens of thousands of individuals, many of them unknown and not listed in any repository. I have contributed some software to the open source arena, and others have taken parts of it and reworked them for their purposes, and so forth. By the time it reaches me, I cannot hope to list the provenance information associated with the widely used GNU C compiler, the components of Perl that I use, the version of Lisp I use, and so forth. The cost of doing such a traceback would be extraordinary and of almost no value. This also ignores intentional subversions placed in mechanisms by government agencies, which we now have evidence is endemic, and which cannot be reasonably traced to origin.

The only alternative for having provenance is to create it all from scratch myself. The cost of this would be prohibitive, and of course you would likely have more bugs with fewer eyes examining and testing it. If we did somehow manage to create the fully vertical company with low prices and high volume, we would likely become a monopoly and be broken up by government. Look at AT&T as an example. Don't claim that their phone service was better than what we have today. In some ways it was much more reliable, but in many ways it stifled innovation.

In today's cost competitive environment (yesterday was at least as competitive, but we like to think it is tougher and we are smarter), this would drive prices too high to win any sales. We cannot trace supply chain through most vendors because they won't support it. Proprietary advantages come to those who don't disclose and liability is increased by those who do.

How about defects?

We would like our software to be free of defects. But what that is, nobody has really explained. Normally, we may think of a defect as something obviously wrong. Like a wheel that falls off the car as we drive. But this assumes normal driving, not driving off of cliffs. So what is normal usage for software?

If we think about normal use of a car, the operator is expected to operate the vehicle safely. This means not driving it off of cliffs, but it also means not driving it though gravel and rock quarries all the time (unless it is sold as an off-road vehicle). If we drive a car on normal roads at normal speeds and do so with reasonable care, and if we do regular maintenance at the schedule identified in the owner's manual, we should expect the wheels to stay on. If they don't, then this is a “defect”.

Software is the same way. If the software is “Internet” software we would expect it to handle the normal Internet environment. We would not expect a consumer grade product to keep running under core-of-the-Internet loads, but under the normal loads of a consumer.

What environments?

This brings us to the question of the environment for a software component or product. It has everything to do with how it is sold and what is stated about it. Here are some environments and some notions about what we might reasonably expect:

- **Private use:** A private user operates in a commonly used operating environment sold as it comes from a retail store. They run through an Internet Service Provider (ISP) using a network address translation (NAT) gateway on a private (non-routable) IP address, or when away from home, on the local WiFi (same setup) or over a cellular network (same setup). They run any free software on the Internet from anywhere, download anything, run applications from banks and retailers, and friends and relatives use their computer sometimes.
- **Small business use:** Like private use except they predominantly run software the business needs to operate instead of freely running anything from anywhere anytime, and most computers are used by workers only.
- **Industrial control use:** Control systems are operated in closed environments dedicated to the function of the process they are supposed to control. They have semi-custom configurations using special-purpose ICS hardware, custom versions of ICS software, non-standard commercially available operating environment configurations, are designed, configured, and integrated into the physical environment by engineering professionals in the specialized field, and tested under identified operating conditions for safety and reliability. They only run the software necessary for the tasks at hand. If connected to the Internet, this is done indirectly through proxy servers, firewalls, or higher surety mechanisms that are customized to the risk and need, well managed, and carefully watched. Only authorized and properly trained plant operators and systems integrators interact with these systems.

There are, of course, many other specifics and environments to be considered, but you get the idea.

In many ways, the requirements for a private use system are far higher than those for an industrial control use system, because the environment of the ICS is far more controlled. Most ICS systems don't have substantial privacy requirements, are physically secured, have trained operators with limited actions to undertake, and are in environments that rarely change except in very well understood ways. At the same time, ICS environments need high reliability and availability, integrity, and use control. Thus, while we can update most private use and small business systems all the time without great problems arising, ICS environments cannot continually be updated. There are many other such things to know about these environments, but I digress.

What can we really expect?

“Defect” free products under malicious attack have never existed. We should not expect it of modern software or systems. We all know that under nuclear attack most everything will fail. Is this a product defect? Are all vendors everywhere supposed to build everything to withstand nuclear attack? Why should all software vendors expected to build all software to independently withstand arbitrary acts of unknown 3rd parties? What constitutes a "known" vulnerability? To them? To the public at large? To the government agency that planted them?

We expect too much of the software and information systems industry. But at the same time, we don't expect enough in select areas.

We should not expect:

- **Perfection:** We won't be getting it any time soon. And if we did the cost would be such that the vendor would go out of business because almost nobody would buy it.
- **Unlimited environments:** Everything that does anything useful only does it under limited conditions. This is not going to change.
- **No updates ever:** Even the White Glove version [3] we have been running since the 1990s years is finally being updated. Not for security, but for all the new functionality...
- **Always on them:** It's not reasonable to believe that everything that happens to you is someone else's fault. You have to be responsible to act reasonably as well.

We should expect:

- **Suitability and fitness for purpose:** The things you buy should work and continue to work the way they do now for a long time to come. They should not break under normal use for the purpose they were sold for. If they do, a recall should be required, liability attached, and it should be expensive for the vendor and free to the user.
- **Environmental specifications:** People who sell software and systems should specify the environmental requirements associated with normal and reasonable use. The details should be there, but common terms like “home use”, “small office use”, and “industrial control system use” should be clearly marked and details available prior to sale (or returns guaranteed).
- **Rare updates and never on an emergency basis:** Anything suitable for purpose should not have to be updated often. We can (and still do) run White Glove form 1999 today, and it's still reasonable safe for the purpose it was designed and offered for. In any case, no update should be required immediately because any features that cause such conditions should be readily disabled by the user after warning to assure safety while the recall takes place.
- **Responsibility properly shared:** The seller and buyer should share responsibility.
  - **The seller** should build things suitable to the purpose and support their use for that purpose by identifying it clearly and in adequate detail.
  - **The buyer** should understand what they are buying and not try to use a home use product for industrial purposes.

Summary:

Do we require that cars be safe from missiles? That they operate underwater? Are safe when driven over cliffs? How about if drivers don't do those things? Software and systems should work reasonably will in environments where they are being reasonably. Asking them to work properly when used improperly is not reasonable. Vendors should clearly mark reasonable uses of products and buyers who use them for other purposes should do so at their own peril.

3 White Glove Linux was jointly developed by me and Garrett Gee in the late 1990s and has been running continually since then directly connected to the Internet without modification or break-ins.

