Controls and Sensors Project Question List (Version 6.0)

Andrew Anselmo
Clipboard Engineering
www.clipboardengineering.com
(To reach me; e-mail first name at this website)

This list originally evolved out of a checklist meant for selecting a few unique pieces of hardware (a Programmable Logic Controller (PLC) and the corresponding Human Machine Interface (HMI), and some data acquisition systems), but has morphed over time to become a more generic question list on what you want to do with your project *as a whole*. It seems obvious, but **systems level thinking** (pulling back on the zoom lens) is critical to ensure the success of any project or endeavor.

It has gone through a few iterations, based on my experience, and the experience of other engineers. Feel free to share with anyone (please keep the attribution), especially with people trying to sell you things. They should be able to answer every one of your questions. If anything, this document will help you bring up issues before you buy your first piece of hardware. If you do have something to add, let me know, and your name will be added to the list, if you would like.

Technical Details

1) **What do you want to do?** What is your system supposed to do? What do you need? Analog input, output, digital input, digital output? Wireless transmission of data? How secure must this all be? What is the needed resolution of signals, the speed of acquisition, the input and output types (relay, DC, AC)? Do you need motion control (stepper motor or servo; what are the sizes of the motors; what are the speed and torque requirements)? How many degrees of freedom do you need? Do you need pulse width modulation (PWM) for the control of motors or heaters? Will any temperature measurement (thermocouple, RTD, thermistor?) be done? Will you need serial (RS-232 and RS-485) communication? Modbus or other industrial protocols? Vision system integration? How many ports for each protocol? Do you need to have a keyboard, mouse, USB keychain drive storage? What is the bandwidth of your signals (both input and output)?

2) **Cost** – How much for the hardware, the software, extra modules, support contracts? This can depend on the number of systems you want to build, and how flexible the system is supposed to be. *This is a big topic, and has to be measured against all of the other items mentioned in this document. See General Notes at the end.*

3) **Expansion** - Is the system expandable? How many IO points can you have? Sometimes there are limits.

4) **Programming - Support**- How difficult are the devices used to modify and program? What sort of support is there? How long has the system been in service? How fast can someone help you out? Do they have a local bulletin board where other users of the system can exchange code, ideas, bug reports? Who will maintain this code?
5) **Programming - Languages**- How well documented is it? Are examples available? What languages does it support (ladder, structured text, and C-like languages are usually the norm for PLCs)? Sometimes the programming is graphical (Labview, Eurotherm iTools), and can be difficult to follow for large programs.
6) **Connectivity** - USB, serial, Ethernet? Modbus (TCP or RTU)? How do you set these systems up? How many different vendors will be involved in this system? If you have one PLC vendor, one HMI vendor, one motor vendor, etc., there may be issues in connectivity and interoperability.
7) **Many Machines, Small Changes** - If you are going to build 10, 100, or 1000 of these, and there are small recipe changes or differences (for example, offsets, calibrations), how will you keep track of these? Machines should be easily and individually identifiable. If each machine has an IP address, can you set this on a per unit basis, or does it need to be programmed in? A small point, but for production systems, you don't want to be changing code for each IP address. If machine-specific information is needed, it should be readily identifiable.
8) **Code Format** - The code itself - is it stored in a proprietary binary format, or is it in text files? When the code is purely in plain text, it makes for easy comparison with free and third party tools. When it is stored in binary code, this can be used as a security feature.
9) **Comparison tools** - How difficult is it to compare two different programs? This is critically important when things get complicated, or when you have different versions of the same code. This can sometimes be tricky with graphical programming interfaces.
10) **Hardware Security** - How good is security for the unit? Does the programming tool require a dongle, or does it require just a registration code? Dongles are a serious pain, because if you lose it/forget it, it is a real hassle.
11) **Hardware setup/software setup** - Does the system automatically recognize hardware? Can you separate the hardware setup and software setup, if you need to modify things down the road? How easy it is to set up on different machines (Macs, Linux, Windows)?
12) **Variables and Defaults** - For the PLC, can you upload/download variables separately from the code? Some systems allow you to do this. This is good when you need to clone a machine, or backup a system.
13) **Recipes** - Do you need to read/write recipes, or do remote data acquisition or updating of systems? B&R has a nice way of doing this via a tool called PVI Transfer; a very powerful way of getting data in and out of the system. Other manufacturers have similar tools; it is important that the recipes be edited offline, with a standard text editor!
14) **Databases and Data Recording** - Getting data in and out of the system - how difficult is it to hook up the PLC to a database? Or dump data to a file? Do you need to spend extra money to get the tools for this? Are there sample programs you can look at? Do you need to look at data in realtime graphically? Most systems have the capability to dump to CSV files, or SQL databases. Once the data is there, how can you easily extract it? There are lots of database tools out there; you can also use other things like Python to extract data from the database for further processing.

15) **Worldwide Support** - If the system is going overseas, what is their worldwide support like, for both parts and technical support?

16) **Integrated Development Environment (IDE) - PLC Simulation** - For the integrated development environment (IDE), do they have a simulator for the PLC, or for other equipment?

17) **Code Documentation** - Can you print out/export the code easily? Sometimes you can do comparisons of code this way. Can you print out a list of variables and their values?

18) **Math Libraries and Functions** - Check out the math library. Does it have the functions you want? If you have some obscure function or logic needed (trig functions, lookup functions, etc.) can the system provide that? You may need matrix inversion, obscure functions you haven't seen since calculus class, or other mathematical tools. If you have function/subroutine capability, you can 'roll your own', but built in tools are preferred.

19) **Changing code 'on the fly'** - If you have a running system, and you need to update it, can you do so? Some systems are a bit more robust than others, and download new code pretty quickly. Others require to stop the program, download the code, and it sometimes takes a long time, which can kill the process you are running. Might not be so bad for big time constant systems with large thermal masses, but sometimes this can't be done.

20) **Memory and Program Size, Language Capability** - How many variables can you use; can you use arrays? Multidimensional arrays? How well is the programming language implemented; can you pass by reference, pass by value (for functions); return more than one value from a function, etc.

21) **Overriding Values/Debugging** - In the PLC IDE, are there tools to override values, or at least see values? Check out their debugging tools.

22) **IDE - Use and Documentation** - Get the manual for the system and the IDE, and see how they work. Do they have search and replace functions? How well is the IDE implemented? If you go to another computer, what files have to be transferred? What happens if you have special routines, icons, etc.? Where are they stored?

23) **Storage and Memory Locations; Battery Backup** - where is the program stored in the system? On a CF card, or in some other memory? Most PLCs have a 'low battery' indicator; check to see if it has one.

24) S**ecurity of Code** - if you download the code, does the system strip out comments? Once the code is downloaded, can you extract it from the machine? Some people want this feature, some people don't. Some systems give you the option of doing both (i.e. Red Lion/Eurotherm Penguin).

25) **Updates** - how are they handled? Do they have release notes for their IDEs? Can new firmware be downloaded to the PLC if necessary?

26) **Hardware - Cabinets** - how does it look, and handle? Do you need enclosures for their I/O points? Do they have ready-made cabinets? Does it have any special connectors unique (and probably costly) to the specific vendor?

27) **Integration With Other Specialty Devices** - Will the system integrate with other elements in your workspace? Do you need the PLC to talk to something like a high precision DVM, or other exotic instrument? Usually, if the systems have serial connectivity, this is pretty much guaranteed, but you need to figure out what you want to do. Modbus TCP and RTU are also other classic standards, more for things like motors and industrial instrumentation.

28) **Web Browsers** - Do you want the system to serve data over the web, viewable by a simple browser? Some systems (more on the microcontroller side) can be even programmed via languages like Python.

29) **Open Source** - Do you want the system to be open source, or at least open source friendly (that is, have your end users modify things)? The Arduino UNO, Mega, and DUE have some pretty impressive capabilities, but may not be industrially bullet-proof as standard PLCs.

30) **Data Acquisition - Ready Made Tools?** Do you want basic data acquisition software to work "out of the box?" Some PLCs and data acquisition systems have great tools for data acquisition, but sometimes they have limitations with regard to storage of data, the amount of time they can run, and how they format and store data. Check to see what they can do!

For HMIs, the same sort of stuff applies, but you can add in:

1) **Ability to log data, create web pages** - How difficult is it to remotely view the screen? Remote access allows you to take screen shots easily, or remotely access your machine from a networked machine. This is a double edged sword, of course!
2) **How difficult is it to interface with your PLC?** Can it share address space, or at least load up a list of the PLC variables?
3) **How difficult is it to build screens**? What happens when you have lots of data? Aligning columns, mapping variables to input/output? This will impact the final look of the system.
4) **What do you need in the HMI screen?** How big (or small) can the screen be? Can you go from one size to another without changing the program much?
5) **Features** - Check out the touch screen features, and find out if the screen works in your environment (i.e., can you work it with gloves on?)
6) **Connectivity** - Does it connect to the PLC via Ethernet/Modbus, serial RS-232/RS-485, or another protocol?
7) **Alarms and Events** - How does the system handle alarms, security, event logging? Some systems can record every keystroke or variable change, which is excellent for process documentation.
8) **What is the HMI programming like?** Is it just an interface to the PLC, or does it have some logic handling capabilities? Usually, keeping the logic on the PLC side for clarity is a good idea, but sometimes, a bit of flexibility on the HMI is nice, and also needed.
9) **Graphics** - Do you want X-Y plots, bar charts? Some systems are very flexible, some are not. Do you want to see pictures, or display PDFs, or manuals?
10) **Color/Backlighting** - Do you want color? This is standard these days, but the depth of color may affect the way you build you screens. Is there backlighting? How is the view angle? Can you see the screen easily off-axis?
11) **Environment** – Where will the unit be located, and what is the physical environment? Heat, humidity, dust; will it be in a washdown or sterile environment?
12) **Security** - Some HMIs can record every keystroke - great for knowing if someone is fiddling, but also good if you forget to write things down.

13) **Don't be enamored of the HMI** - Sometimes, physical switches, buttons, and joysticks are better choices for inputs to the system.
14) **End users** – Who will operate the unit? Will you design for users who may have be visually or auditory impaired, or color blind?

Non-Technical Details

1) **Not All Companies Do Everything Well or Cheaply** - Some companies make great software for one thing (Matlab), but may fall down in the automation area, or great automation software, but fall down in database connectivity. Check to see what they've done (references - see below).
2) **References** - Ask your vendor for references - how many people have used their hardware/software? This is *critical*. See if you can talk to those companies "off the record"; many won't bad mouth their vendor, but will give you a good sense of how many problems they have had. Ask them this one question - "Would they use this/specify this system again?"
3) **How Important Is This Hardware To Your Vendor?** How many in-house developers do they have working on this project/system? This gives you an idea if this is a sideline or a main thrust of their company.
4) **Upgrades** - How often are patches released? Are these patches free forever, or is there a maintenance contract? Beware of automatically upgrading; sometimes, other things break after something else has been fixed. Always ask for detailed release notes, if possible - *n.b. Don't upgrade anything before a demo to investors or contract monitors.*
5) **Beware of 'bleeding edge' technology.** Stuff that has been around for 5-10+ years may not have the slickest interface, but have been tested in the real world, by real engineers, solving real problems.
6) **Good Support Will Be Rewarded** - When discussing with vendors, make sure they know you talk to other engineers. Tell them systems you are happy with, and systems you are not happy with. Knowing you will extol their virtues to others, or hammer them informally can be a powerful incentive.
7) **Help With Your Project** - Asking help for one-off projects is usually more difficult than asking for help when you are going to buy 10, 100, or 1000 units over a long time. You may want to quote the upper end of your build projections, so they will take you more seriously. Talk to sales reps representing different vendors, and get a feel for their product as well as their competitors. Don't be afraid to ask "Why should I buy X from you, instead of Y from the folks down the street?"
8) **Demo Units** - Get demo units/software whenever possible. Usually, you can get them on loan for a few weeks to evaluate things. The engineer who demos the system should be able to get you set up quickly. If not, there may be issues down the road.
9) **Know What You Want and Write It Down!** - Write 'requests for quote' so you have your specifications in writing; this helps get your own 'house in order', and gives all vendors competing a common plan to work from.

General Notes

1) **Always think about what you want to do, and what is your final goal**. Ask yourself "Has this been done someplace else? Can I buy it off the shelf?" Can part of this solution be off the shelf, with modifications?
2) **Get your designs reviewed by someone else -** It is great to have a fresh set of eyes on things. Preferably, by someone not in your immediate group; even better if it is someone outside your company, who isn't in any hierarchy or who has any interpersonal issues with you or your team. A few short hours of someone from outside can save weeks or months, when they point out potential problems you have overlooked.
3) **Beware of the Not Invented Here** (NIH) syndrome. You don't need to reinvent the wheel.
4) **Remember the general engineering mantra -** "Better, cheaper, faster - pick only two." If someone thinks you can get all three, they are handwaving the fact that one of those items will be more expensive!
5) **Think about overall cost** - Many times, a 'cheap' system is cheap upfront, but requires lots of engineering/programming time on the back end. This is especially true for systems that are low volume production systems. The cost of controls may be minor compared to the cost of the rest of the system.
6) **Document, document, document!** A good system that has documentation can be debugged, upgraded, understood and fixed; a slick system with no manual can be a nightmare to maintain, use and duplicate. For hardware, take lots of digital pictures; for HMI screens, take lots of screen shots. Documentation should be tightly coupled to the system; if possible, document within the code itself.
7) **Backups -** When writing software, make sure you have a good version control system, or other way of being able to 'go backwards' and get to a working version of code.
8) **Use version numbers in your software** – This is so you know what version is being used in the machine. Display this on the HMI screen someplace. The date can make a great version number; YYYYMMDD.
9) **Quality** – It is remembered long after the price is forgotten.
10) **Do the hardest thing first** – If there is a difficult element in your system, do that first, and make sure your system can do the thing you want. If that requirement is collecting data at a rate of 10 kHz, make sure that it can be done. If not, all else is moot!
11) **Use test widgets** – A few Arduinos can simulate a device with a serial port, with realtime responses, connect to really cheap sensors, and act as a stand in for more complex or expensive equipment. Don't laugh; an Arduino with an Ethernet shield can look like a Modbus device, be a webserver, and interface with I2C or SPI sensors, which some PLCs cannot work with natively. There are industrial versions of the Arduino, if you need them. At the same time, one shouldn't use them in safety related applications, or where life and limb are at risk.
12) **Be careful where you get hardware** – Hardware is seemingly ubiquitous, especially with Amazon and Alibaba, but getting the same hardware six months later may be difficult or impossible. Places like McMaster are more expensive, but do have things in stock.

13) **Supply chains are fraying** – If at all possible, design using parts that can be interchanged. You may save money in the short term, but if you need two types of motor controllers, from two different vendors, you will need double the spares.
14) **Not everybody is an expert at everything –** As much as smart people like to think they can handle everything, it is sometimes best to partition a project into different sections. Yes, mechanical designers can program PLCs, and yes, systems people can create wiring diagrams, and even do PCB design, but when things get complicated, it is better to hire experts in those specific areas. You don't have to hire an individual person for each part of the project, but at some point, things can get out of hand for one person, both as far as time and expertise. There will be some overhead in communication between everyone, but the net result is that your final project will be professionally done in all areas. Any consultant who says they can handle everything should be a bit suspect.