Introduction

Most manufacturing and mining businesses will have the words “continuous improvement” written into their business strategies. Improvement can be in many areas including:

- Increased customer satisfaction.
- Increase in output per unit cost. (improved efficiency)
- Improvement in quality of produced product.
- Increase in range of products produced to suit market demands.
- Decrease in equipment down time.
- Process improvements through the adoption of new technologies.
- Increases in safety. (Zero lost time through injuries)
- Increases in employee morale. (positive attitudes help businesses succeed)

Some specific processes and operational equipment will never require modification. Their control system design will be adequate for the intended purpose. Some systems are intensively designed and should not be modified, without strict change management procedures. (i.e. petro chemical) Other systems may require continuous modifications to suit a changing environment, process, product line or reliability. This article is about the process control that is expected to change. It is based around PLCs, but could easily translate to any programmable control system.

The most common tool for process control system modification is “software”. This path is chosen because,

- it is the least expensive option.
- it can often be modified on line or with minimal downtime.
- it can revert back to the previous version if the upgrade does not measure up to expectations.
- it can have its capabilities extended without a complete re-build. (if it is well written)

Larger businesses may have the ability to employ their own **P**rocess **C**ontrol **S**ystem maintenance personnel. These may be at all levels from tradesmen through to design engineers. This offers the opportunity to develop PCS standards and an integrated system, but it is not always taken. Smaller businesses will usually sub-contract process control personnel on a part time or project basis. With a variety of process control companies offering specialised services, it is important that the work done by these companies is homogeneous and integrated. It should not be a compilation of clunky bits and pieces.” How will the owner or manager of the business ensure that this is the case?” Answer: “By insisting on **maintainable software**”.

What is Maintainable Software?

In theory all software is maintainable, however the degree to which the code is maintainable is the point of discussion here. An assessment into the degree of maintainability of a given piece of software

may conclude it would be cheaper and/or less risky to rewrite the code than to change it. Such code, for all intents and purposes, could be classified non-maintainable. The original investment is null, and void.

Attempts at deriving a maintainability index for code have been sought, based on simple lines-of-code assessments to the more elaborate Halstead complexity measures. These techniques are quantitative. It is our belief that a qualitative approach is more successful when applied to PLC code.

So what are the qualities of maintainable software?

1. **Structure**. The importance of structure can never be underestimated in making software understandable by others. Process control is a series of functions that combine to make a control system. (functions with glue) The categorisation of functions needs to be carefully planned at the start of development. Think in terms of module functionality (the task it needs to perform) not in terms of logical outcomes. Hierarchical flow charts help to visual the functional layout. Software should be designed top down, with high level functions gradually being divided into smaller functions that can then be developed. The structure should mimic the process functions. Develop separate code for each piece of equipment. This makes it easier to narrow in, if modification is required later. It also improves the speed of fault finding.
2. **Well Defined Functions**. Functional specifications, in plain English, with explanatory diagrams to aid others in understanding the purpose of a section of software. If requested to modify it, there is less chance of mistakes if they have a good understanding of what it was originally intended to do. Anyone in the industry should be able to read the functional specification and understand the purpose of a piece of software even if they cannot read the code. Functional specifications are the link between the process designers and the process control software developers. They are also an educational tool in helping operators, maintenance staff and managers understand the operating parameters of a piece of equipment. If you understand its function, you can judge if it is living up to its expectations.
3. **Standard Naming Conventions**. Agreed naming, used as labelling for equipment and matching labelling within the software. Plain English labels that mean something to everybody. Languages can have multiple terms for the same object. This leads to communication confusion and mix ups. A clearly defined naming convention allows operators, maintenance personnel, software developers, machine designers and management to all communicate reliably.
4. **Commenting**. Every data tag requires a functional description. (what is it, what does it do) Every line or rung of code requires a comment as to its purpose in the overall function (even if it is obvious). When developing, it is easy for the developer to get something clear in their head and produce a run of quick uncommented code as they concentrate on the relationship between bits, variables and pointers. The chances are, when they revisit their own code in 12 months, they will have no better understanding of the code than someone who is seeing it for the first time. There is no problem with the quick production of code through this method, as long as at the handover, it is fully commented. Commenting should be treated as if you have a first year apprentice reading the code for the first time. Each comment should be a training

explanation. This may initially seem time consuming, but a relatively small amount of time explaining the purpose of code now, will save a large amount of time in reverse engineering and possibly equipment damage through mistakes at a later date.

5. **Data in Standard Engineering Terms**. Early programmable controllers had limited data structures and mathematical capabilities. Data was often treated in its raw value to prevent rounding errors. (this may still be the case in some smaller PLCs) Modern PLCs, with maths co-processors, offer little penalty for working in engineering values with floats. (or 16, 32 bit integers for larger numbers) From a maintenance perspective, following numbers with meaningful values is much easier than following raw data. There is less chance for mistakes if the numbers are understood. Conversions to engineering units may be achieved in the input cards or with scaling blocks as the data is read. (Scaling blocks on raw data in viewable code are probably a little easier to understand) Data is then used throughout the code in engineering units.
6. **Standard Modules**. If common pieces of equipment are used in a process or in other parts of the plant, then there is no reason not to use common code to control them. Even if there are small variations in the exact nature of similar processes, it is still viable to produce common code that has inputs that switch certain functionality on and off. With staff turn overs, standard modules offer maintenance personnel a learning advantage. It also allows the software maintainer to tweak standard modules to eventually make them bullet proof under all failure conditions. As improvements are made and tested on one piece of equipment, it becomes a simple job to roll them out to others.
7. **Ladder as Power Flow**. Although it is not essential, most process control programmers will have some computer based education. They will understand the workings and operational features of microprocessor based control and associated high level languages and their compilers. There may be a tendency to develop code that is optimised for minimum memory usage and compiler efficiency. (show off how well you understand the microprocessor functions) This is not desirable for maintainable software. Despite its opposition from computer scientists due to its inefficiencies, ladder based code is still in extensive use throughout industry. Many other languages have been proposed and are available within the PLC suite. Ladder is still in common use, because it is the most readable by everybody from apprentice to master engineer. It is the easiest to follow because of the concept of power flow. The visual representation of power flowing through a series of switches to drive an output is the easiest to comprehend. Good software developers need to be mindful that this is the reason they are developing in this language. (and not in “C”)
8. **Operator Defined Batching or Recipe Control.** With this technique operational inputs, step conditions and outputs are defined in tables that are directly entered from a HMI panel interface or a SCADA system. This technique allows changes to be made without the need to alter PLC software. It also has the advantage that the process functions are on continuous display.

Why do we need to maintain software?

Software maintenance is driven by a businesses need for continuous improvement, as stated in the introduction.

IEEE standard 1219 defines software maintenance, as the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.

Software maintenance can be classified as (ISO/IEC 14764):

- Corrective maintenance: Reactive modification of a software product performed after delivery to correct discovered problems.
- Adaptive maintenance: Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment.
- Perfective maintenance: Modification of a software product after delivery to improve performance or maintainability.
- Preventive maintenance: Modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults.

Below are typical software maintenance functions:

1. **Improvement in monitoring and reporting functions of the process**. The first stage in process control improvement is the measurement of all the input variables and the resultant outputs from the process. This is usually achieved via data logging and then displayed with trending packages. It allows managers, designers and analysts the ability to view relationships between the influences on a process, the processes response and the resultant output. Improvements in measurement and an increase in the number of variables being measured can show surprising results, that offer pathways to improvement.
2. **Improvement in control algorithms**. As more variables in the process are better understood, modification to control algorithms can “fine tune” performance.
3. **Experimental testing of new ideas**. Lateral thinking will often produce a whole different concept in how a process might be controlled. With careful coding, it is possible to build an alternate method that can be switched in to replace the old method with a single bit. This method allows development of new ideas during maintenance or slow periods with the ability to switch back to the old system quickly and continue normal production.
4. **Expansion to include additional or different equipment and materials**. Equipment upgrades nearly always involve the expansion of the control system to include new measurements, additional outputs or different control sequences and algorithms for different materials.
5. **Improved fault and failure diagnostics**. Diagnostic software incorporated into the controller can improve mean time to repair, by quickly identifying problems for maintainers. As more is learnt about a process and its failure modes, more diagnostic software can be developed.

6. **Improved reliability**. Reliability or robustness in process control software, refers to its ability to detect and act appropriately to equipment or other failures. Development in this area is associated with orderly process shutdowns or bypasses, without the system going haywire. It may also include automatic switch over to back up systems in critical situations.
7. **Improvements in predictive alarming**. Predictive alarming has the potential to greatly reduce down time and rate losses. This normally incorporates increased monitoring by the process control system, with an associated rate change or variable value software warning system. It may also monitor for a particular sequence of events which have the potential to cause problems. These flags may be acted on immediately, by an operator changing settings that will avert a shutdown, or may be an indicator that programmed maintenance is required.
8. **Improvements in safety monitoring**. Human protection from injury is paramount in any business. As technology advances, safety systems are also advancing. Process controller systems are generally not part of a safety circuit, however they are used to monitor and report on the safety systems and therefore are required to be updated.
9. **Improvements in statistical data for business analysis**. Process control systems are usually involved in the collection and collation of raw data for business analysis and other high level tools. Given the amount of raw data in a PCS, more of this data is being filtered or manipulated to better suit business requirements, rather than just the PCS requirements.
10. **Improvements in the Human Machine Interface** (morale boosting). Even without process improvements, significant productivity gains can be made by better connecting equipment to operators. A well thought out interface providing operators with what “they want” to optimise their performance, will be an improvement to any business.
11. **Temporary work arounds and bypasses**. Often small equipment failures can be tolerated by a process with only a minor loss in time or rate. At the discretion of authorised personnel, temporary software modifications may be needed while repairs or replacements are completed.

The largest problem with any software change is that it has the potential to adversely affect some other previously working function. This comes about because the software modifier has not completely understood the functions of the code being modified or some of its related sub-functions. This problem is exacerbated if the software is not easily maintainable.

There are obvious business advantages in insisting on maintainable software. Historically, clients have been naïve when it comes to negotiating with suppliers for the quality of the software they receive. As long as it functions as specified at handover, they are satisfied. Some software providers are ethical and hand over equipment with the software developed to the best of their ability. Others may not, with budget pressures or sometimes epistemic arrogance forcing short cuts in quality. In the industry, there exists a black box mentality towards the “propeller heads” that sit in a room and produce control software. This should not be the case, with the principles of good software development falling into line with any other principle for good business management.