The Role of Automation Systems in Management of Change

Similar to changing lanes in an automobile in a winter storm, with change enters risk. Everyone has most likely experienced that feeling of changing lanes in bad weather where you find yourself weighing the risk vs. the benefit that the new lane provides. The same principle can be found when considering making changes to control applications and automation configurations in an industrial plant. In order for the change to have a positive impact as was originally intended, the risk that is associated with making the change must be mitigated. There are multiple ways that risk can be reduced such as following good management of change processes and procedures, personnel training programs and certifications, and having a safety minded corporate culture. However, one of the best assets that those responsible for implementing change has is the automation system itself.

Change is part of life and business
First let’s examine why a change is made to a running facility due to the fact that the first question someone will pose is “If there’s increased risk, why make the change?” or “Isn’t it better to be safe than sorry and not make the change?”. While these statements are true, there are benefits that justify making changes provided that every effort can be made to reduce the risk that they incur.

Incremental changes to process automation systems are required today more than ever due to regulatory changes, expansions or additions, optimization, and product variances. In some industries, it is acceptable to have frequent shutdowns to safely incorporate these changes. Adversely, in mission critical, continuous processes such as oil and gas, petrochemical, and power shutdowns are few and far between. In most cases, major outages in these types of process applications will only occur every 3-6 years, thus changes to a control system’s configuration are a necessary and acceptable practice. Below are just a few examples of changes that occur in a running facility:

- Process modifications
- Production optimization
- Plant expansions
- Enhanced safety measures
- Implementation of new regulatory or environmental requirements

Any of the above can greatly impact business metrics which can justify making changes to a running facility. For example, implementing a change in a chemical facility that optimizes production can increase production or result in a higher yield of a certain grade product. Implementing this change makes the company more competitive, efficient, and profitable thereby providing justification. If this
change was only allowed to be made during a planned shutdown or outage, the opportunity lost could mean million in lost revenue depending on what is produced.

Another example of where change could impact a facility is the implementation of changes that are necessary to comply with standards, best practice guidelines, or regulatory requirements. Regional legislation may require the addition of monitoring, controls, and reporting in order to comply with new standards meant to reduce the impact to the environment. These improvements normally have to be made within a certain timeframe which can impact the cost of implementation. If a facility has to shut down production in order to make the change, then the cost of implementation could increase to millions of dollars. Of course, if the changes are not made until a shutdown or outage, there may be financial penalties or possible harm to the environment that could have been avoided.

**Defining necessary changes**

Now that we’ve identified “why” changes are necessary from a business perspective, let’s examine what kind of changes are made and the possible consequences that can occur.

1. **Operational changes to applications:**

   Operational parameter changes vary slightly from site to site and company to company based on the chosen operations philosophy and functional design of the application. However, these types of changes can generally be classified as modifications made by operations that they have permission to change. An example would be to modify the gain of a PID loop within a predetermined, acceptable range or modifying a low level alarm limit on a monitored process variable that doesn’t have process or safety implications. These changes are usually considered to be ‘low’ risk but are tracked via an audit trail to capture:

   a. what was changed such as the tag name and parameter that was modified
   b. who made the change which is only as accurate as the granularity of user security configured (“john smith” vs. “operator 1”)
   c. where the change was made from usually indicated by a node name (“Console 1”)
   d. when the change was made
   e. why the change was made which, depending on the sophistication of the system or integrated change management / documentation systems, could be a comment added to an audit trail message

2. **Runtime configuration changes**

   Runtime configuration changes are those made to, deleted, or added to, a program or application currently running in a distributed process controller. These changes may require that the modified program replace the current running program or application. The impact that these changes have are based on the architecture and configuration of the automation system. Usually, the smaller the program or application, the smaller the possible impact to other programs and applications, however, the risk may be just as great. For example, a single PID loop could be the program being
modified. The importance of this PID loop and connections other loops, interlocks, and permissives dictate its contributions to the overall risk. Some examples of commonly made changes are as follows:

a. Addition of a new tag or tags, such as a PID loop, the addition of an entire unit, and/or addition or modification of optimization algorithms running in a controller.
b. Modification of an existing control tag such as its range of operational parameters or safety boundaries
c. Modification of interlocks or permissives.
d. Modification of graphics which need to be downloaded to various consoles. Graphics are considered by many to be application code which need to be treated the same as runtime changes as they may impact, both positively and negatively, the operations of a facility.

3. Changes to the automation system infrastructure

a. Modification of execution parameters such as the time it takes for a program to execute or I/O card parameters such as signal types and ranges.
b. Addition, deletion of nodes, controllers, communication, and I/O cards.

Any of the above mentioned types of changes can introduce risk without the proper processes and procedures in place.

Automation systems as facilitators

Automation Systems can facilitate and provide best practice facilitation of the management of change process. In addition to tracking and documenting change through features such as “audit trails” mentioned earlier, modern distributed control, or automation, systems today have many features that minimize risk when implementing change.

1. Catching mistakes before they get downloaded: Whenever humans are involved it is inevitable that mistakes will happen. A value will be mistyped or a tag name spelled wrong. The key is to reduce the chance of these mistakes reaching the running system. To do this, there are multiple tools available depending on the system used. One example is compiler checks / error detection or difference reporting prior to download. A difference report will provide a comparison of the “before” and “after” application code as well as for the operator graphics.

Figure 1: Difference Report
2. Controller implications: Timing is everything for a process controller. In many cases prioritization and task executions may be changed based on the program. Making changes to the program will result in a change to the resources utilized. In some cases, the change may affect the controller’s execution and cause propagating problems for other applications running in the same controller. One example is where making a change to a cycle time could produce unwelcomed results. Due to a change made in the time it takes for an application to run, the controller may become overloaded. Typically this will not be recognized until the download occurs and the controller stops or acts badly.

3. Change verification: During modification of the program, good change management practice usually includes a quality assurance step that verifies that the intended change was the “only” change that was implemented. This ensures that someone else hasn’t inadvertently made a change to the same code that is slated to be downloaded. If this isn’t caught prior to downloading, the compiler detection may not flag it as an error and if missed by the user difference report, the rogue change will be downloaded possibly yielding unexpected results. In some cases, there are ways to compare the application code before and after versions to report on details of the changes made. Comparing the two can reveal that only the intended change was the one that was made. This is also important beyond logic configuration. Graphics are often considered as important as the application logic as they determine what the operator may see in certain conditions. Therefore, it’s important that graphics use the same conversion. Also, it’s important for this reporting to be easy for the non-engineering employees as it’s usually quality personnel doing these checks.
4. Step-wise change introduction: Library Versioning: In many systems in the past, there was only one library of blocks used in a DCS for configuration however, today, some systems have library versioning approach. This means as changes are made, they can be made to a new version of the solution library. The changes made to the new version of the library do not reach the running application until it is pointed to the new library version. This allows changes to be made to the library to be rolled out in steps which can greatly minimizes how much is changed at one time which minimizes risk. For example, if you had multiple units in a facility that was using a certain version 1 of a library and you wanted to make modifications to the solutions in the library, you would create version 2 of that library. Implementation of the changes can then be tested on one unit, then the next and so on, by connecting the different applications one by one to the new library version. The versioned library concept can allow the step-wise change introduction which limits the impact an error will have on the remainder of the facility.

5. Cross Referencing: An important part of reducing risk upon making a change is determining what will be affected upon the change being implemented. Most systems have cross referencing tools to tell where certain variables or process points are being used. Again, graphics is another area in which cross referencing is equally important due to the fact that making a change to an operator display graphics could affect how operators interact with the process.

6. Simulation: Simulation of an application for testing will demonstrate that the application is capable of executing the automation strategy based on the simulation model. The most accurate simulation would be using the actual code and graphics running against either a “soft” controller or in an isolated on-line controller running against live I/O. Simulation serves two purposes, it enables deployment and testing of batch and continuous control applications in an offline environment and training of operators against the system they will actually use.

Impact Analysis– the final frontier in change management

While simulation testing will validate an application’s functionality relative to the its requirements, it cannot show the actual differences between the new and the original application in order to unveil any dynamics that might exist as the consequence of the new application calculating different outputs. For example, upon the execution of a newly downloaded application or program, the new application could result in closing a valve where the existing one would open the valve. The result, could potentially have consequences such as poor performance, shutdowns, environmental impact and increased risk to safety of plant personnel.

Therefore, true impact analysis can only be accomplished where the modified application or program is downloaded to the running controller in parallel with the original and, using actual inputs, the user can “evaluate” the resulting differences, if any, between the “evaluation environment” and the actual, running “production environment”. Though this feature is not common in all automation systems, it is
a powerful tool that provides two distinct benefits. First, it significantly reduces the risk associated with making application or program changes in the running process and second, it improves overall efficiency by avoiding production stops, poor optimization, and costly downtime.

A feature of ABB’s System 800xA was developed in collaboration with DOW called “Load-Evaluate-GO” or sometimes referred to as “LEG”. This Load-Evaluate-GO feature enables the user to add programs or modify configurations and then load the new application into the running controller in parallel with the application actually running production.

Using actual live inputs, the “evaluation” version of the application calculates the outputs that would go to the field if it was the actual production application. The two environments can then be compared directly to the actual running application via difference reports of signals, alarms that may be generated as well as an operator interface view that is generated separately from the runtime environment. This parallel universe is called “Evaluation Environment” and can be called up so that the user can see what an operator would see as part of the evaluation.

Once the user evaluates the new program or application’s behavior based on live inputs, they can commit to the new program (“GO”) which replaces the running application OR the user can back out of the process with the original program still running production as it was before.

Because of the very stringent management of change process and its difference reporting capabilities at each stage of the loading process, the LEG toolset forces the user to think about the differences reported in two major situations:

a. Static differences: all the differences between the running code and the code to be loaded are flagged. If a difference is a surprise, load process can be stopped and the situation can be investigated. Also the inverse is the case here: one is expecting a difference and it is not reported. This indicates that a program part that was supposed to be loaded is not incorporated in the load set.

b. Dynamic differences during evaluate mode: even if the static difference test has not identified issues, there is still the risk of calculating a different output value for one or more outputs between the old and new application. Those differences are reported and offer the ability to the user to intervene prior to activating the application such as putting the output of a certain loop in manual or back-out of the loading process altogether.

ABB System 800xA’s Load-Evaluate-GO:
1. The user downloads the modified program to the running controller in parallel with actual program that is running.
2. The user “evaluates” the modified program by comparing output values and alarms calculated using “live” inputs.
3. The user can commit to the new program (GO) or back out and continue editing the new program.
Take, for example, a critical process in which changes need to be made. Even though the changes could be extensively tested and simulated in an off-line environment, using System 800xA’s Load-Evaluate-Go feature could reveal that downloading the new, modified application would have resulted in a particular control valve’s output increasing by 10%. Without the capability of detecting differences in outputs during implementation of the changes, plant operation may have been unintentionally disturbed, impacting the end product quality or worse a plant trip.

**Figure 4: Load-Evaluate-Go**

**Conclusion**
Change is a necessary part of today’s operations in production facilities and therefore management of change is key in reducing risk to the process, business, environment and personnel. Though management of change is much more than features in an automation system, these features can help facilitate management of change in various ways. Compiler checks, change verification reports, the use of libraries for step-wise change introduction, and simulation testing are all ways that most modern systems have available to help management of change process.

However, probably the best way to perform impact analysis involves comparing the modified application or program to the original using the actual live inputs in order to detect potentially hazardous changes to the process. This is not commonly found but has been proved to be an invaluable asset for reducing risk as it detects possible differences in calculated outputs which cannot be found in any other testing scenario. To quote a well known figure “Change ….good ; Fire ..Bad”.