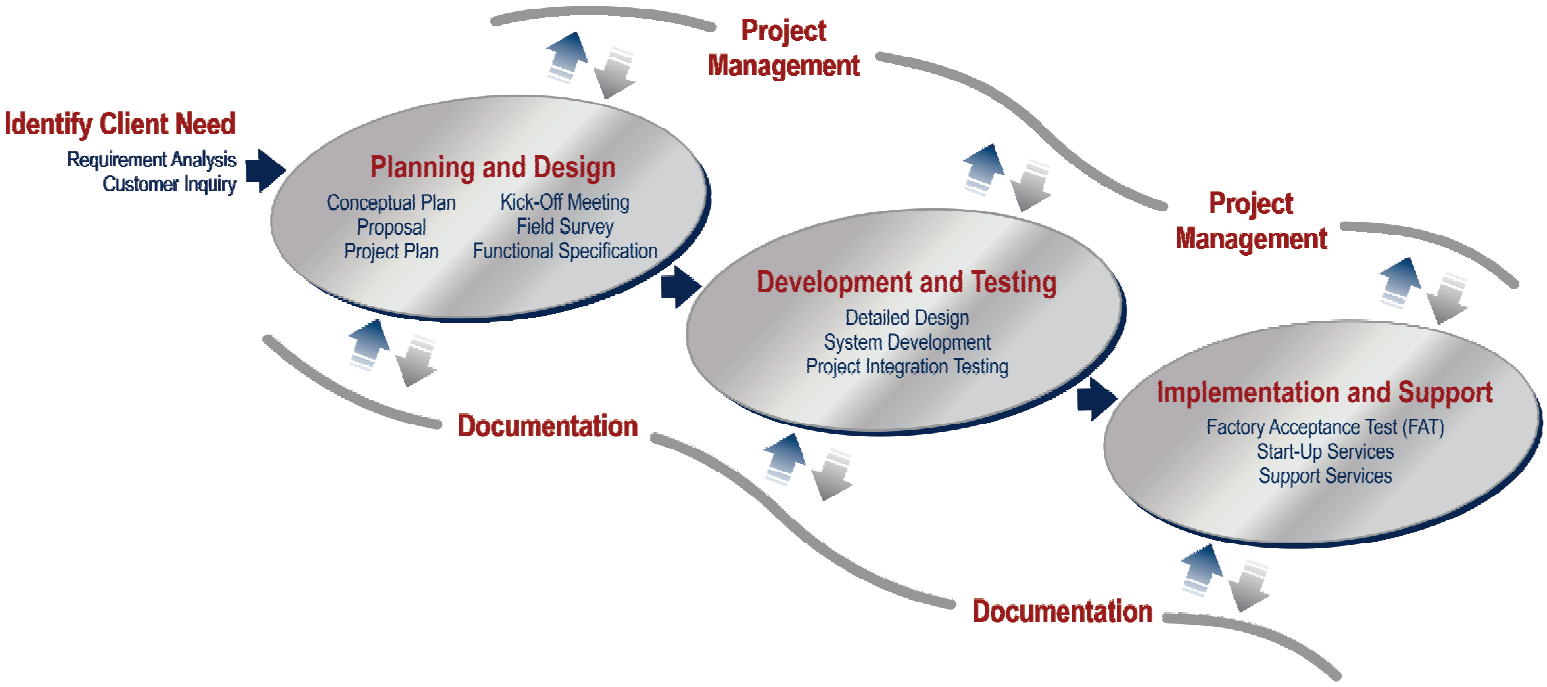
Programming Methodologies and Total Cost of Ownership



INTRODUCTION

If you have ever been involved with or had to fund a new application or an upgrade to an existing application and experienced excruciating pain (cost, duration, delays, …) in the process, then you might be interested in this.

Carpenters have a saying – "*Measure twice, cut once*". It's not quite the same, but planning out how the system should operate and perhaps as importantly *what you want to prevent it from doing* and even more important how to recover from an "abnormal exit" is 'required thinking'. Just for clarification, I am calling an abnormal exit some sort of ungraceful and unplanned stoppage for the system. That could be a power outage or perhaps a drive failure or a chain breakage or an E-STOP or …….

BREAK IT INTO CHUNKS (modular)

Just as you eat an elephant one bite at a time, you also have to *tackle a large solution one sub-system at a time*.

I need to point out that this is not a treatise on "modular programming" in its purest form which is defined as "a software design technique that emphasizes separating the functionality of a program into independent interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality". But this concept of thinking in terms of sub-systems and working to keep system functionality in the same paragraph as it were, does serve to help drive more modular results in real-time programming applications. So with your permission, I'll continue to use the term "modular" even though there is a more refined definition than the manner in which I'm using it here.

Programming Methodologies and Total Cost of Ownership

PLANNING

In this paper, I'm talking about a thought process that is modular and therefore the sub-system operational theories will also be modular. This is different from just addressing needs as you think of them.

So before we jump into the weeds trying to outdo one another in creative mishaps, let's just talk about the need for planning. It is better to write some sort of document that describes what the system is to do which includes how to start it up, how to bring it on line, how to shut it down gracefully than to dive in and start writing code as you think of things you want it to do. If there is some sort of flow control that is going to be required, incorporate that theory for affected control loops. Determine what information is relevant for the various roles being played by operators, supervisors, IT and management. We haven't even gotten to "housekeeping" yet. The thing is, these activities have to eventually happen and it is better to frame your thoughts and plans at a very high level using a high level tool which in my case is a word processor and in English (or at least a mid-west variation of it). Some people like flowcharts others use a spreadsheet. Just pick some tool that is second nature for you to use. Then after having this high level document that clearly identifies your overall system and the sub-systems if applicable you'll have an outline of your plan for how you want to manage your system and the information flow associated with it. The programmer can now concentrate on the various sub-systems and the syntax (the precise lexicon of the programming language being used) to accomplish that plan.

TECHNIQUES

For the real-time control (PLC or PAC), we here at BCI tend to use a technique called "***finite state***" programming. That helps force modularity. The biggest thing that helps modularity is having your operational description arranged in modular chunks. For example, if you have a source of supply that has to service multiple destinations then you might want to think of that "source of supply" as a vendor servicing multiple "customers". How do you want to manage those priorities and how will you divide your attention among them properly so as to avoid or at least minimize disruptions? Is there any particular behavior that you want these destinations ("customers") to follow in order to provide better service to them? Once you get those sort of dynamics figured out for the whole system, then you can start writing the programming code to accomplish those objectives.

TESTING

Don't forget to test as you go! Modular programming will have sections of code that is looking for certain inputs and will be responsible for certain outputs. With that level of modularity you can test that section of code to ensure that it performs as you might expect to include how it might respond to power failures or other abnormal interruptions. The earlier you can catch a mistake, the easier and cheaper it is to fix.

SIMULATION

Then after the entire program is written and incrementally tested, it's now time to put the whole thing together and run a simulation where loading can be applied to ensure that all these modules play well together and that system loads don't expose any timing issues or communication issues. Once you've systematically been through this scenario, now it's time to introduce it to the actual application and once again test out the sub-systems and then gradually bring everything together until you now have the entire system playing together in the real world application. It's always best to do dry runs first and then introduce some testing with low cost product (if possible) before turning it over to production.

Large systems are usually a collection of smaller sub-systems and if treated systematically, can flow quite smoothly.

Programming Methodologies and Total Cost of Ownership

SUMMARY

This may seem like it would add cost to a project, but on the contrary it will save cost if measured all the way through commissioning. That time at commissioning where contractors are there to support the system, operators are standing by and raw material is being consumed is valuable time. The quicker you can get through the commissioning activity and start making finished product, the lower the total cost.

All the best – and avoid "spaghetti code" by applying discipline and a structured methodology.

ABOUT BCI

Bachelor Controls, Inc. (BCI) is a leading provider of control and systems integration solutions to manufacturers with focused experience in the food and beverage, pet food, pharmaceutical, plastics, specialty chemicals, and feed and grain industries.

Batching systems are an everyday occurrence for BCI. Our very first jobs were batching systems. We were applying ISA-88 batching standards as early as 1994.
Founded in 1983, Bachelor Controls has been recognized as a Charter CSIA Certified Member, Charter Rockwell Automation Solution Provider, and Microsoft Certified Partner and is included in Control Engineering's System Integrator Hall of Fame. BCI is a licensed engineering firm based in Sabetha, Kansas; with branch offices in the Kansas City metro area and Memphis, Tennessee.

Ray Bachelor
President
Bachelor Controls, Inc.

Bachelor Controls, Inc.
123 N. Washington
Sabetha, KS 66534
Phone: 785-284-3482
Email: bciInfo@bachelorcontrols.com

Web: www.bachelorcontrols.com