The Almeo Software Development Process

A process should not be a set of rules. At the core of a process, there should be a philosophy based on a set of principles. On what principles do we base our process? They are:

1. Software development is the act of creating and validating an automated expression of human knowledge.

2. The operative word in this definition is "human". Humans are imperfect. Therefore, human knowledge is apt to contain ambiquity, inconsistency and inaccuracy. Often, a client is not even aware of issues with their own knowledge until they see an automated expression of it. At this point, they are forced to confront the limitations of their own knowledge because, at least in some way, the automation fails to work properly. We always begin a project with this assumption:

3. The requirements will change. Succesive software iterations are generally the most efficient and effective way of elaborating the knowledge of a problem.

4. A software project will consist of at least a problem domain and a solution domain.

5. Knowledge is the interrelationship between elements of information within a domain.

6. An ontology is some sort of model intended to map or represent the structure of these interrelationships. An activity that creates an automated expression of knowledge is refereed to as being onto-centric.

7. A development capability should create high quality software, in a timely manner, at a reasonable cost, i.e., faster, better, cheaper. This is often fulfilled largely by creating software systems that are efficent and flexible and by maximizing the amount of onto-centric effort.

8. The extent to which the knowledge structure within a solution domain reflects the knowledge structure within its corresponding problem domain oftentimes determines the extent to which the software system is efficent and flexible.

9. We try not to impose a solution on a problem. The solution should emerge from study of the problem. Many possible solutions can be made to work with sufficent expense and effort. However, developers should be given freedom and support that allows them the create beautiful solutions that are in harmony with the problems they seek to solve.

10. Any automation, utility, or methodology should promote or, at the very least, not impede this harmony.

Type of development work we do most. Although we can do both software and hardware development, we specialize in smaller embedded projects, typically using PIC, ST-7, and ARM7 processors with less than 10,000 lines of code. However, because of various techniques that we use our projects tend to remain rather small even though they may have hundreds or even thousands of requirements.

From the standpoint of our client, the development process look something like this:

1. A customer submits a product specification, often as an MSWord document, that describes how the system or product functions.

2. From this document, we create a requirement set that is entered into a database. From this database, a software requirements specification (SRS) can be generated automatically. This becomes the working document for the rest of the project because it perfectly reflects the requirement set that we are using. Requirements change so this document is being automatically updated through out the project.

3. Next, we design and code the software system. We try to complete the initial release with 2 to 8 weeks. It may be less for a smaller project. This might programmed onto a prototype system. It might also be in the form of a Java mock-up of the functionality if the hardware is not ready or if the control is part of a system that can be simulated more conveniently. We might e-mail the client a .jar file for stand-alone use of the mock-up or the mock-up could be uploaded as an applet to a secure web site so it is accessible to anyone with the appropriate browser.

4. Next we begin development iterations. These will be 2 to 4 weeks. Also, a larger project may be divided into phases. A phase is any small enough so that the intial release can be completed within 4 to 6 weeks.

5. Upon the final release, all work products specific to the project can be turned over to the client. Any additional maintenance is contracted under a separate RFQ.

Knowledge-centic development improves quality and productivity. In 1906, the Italian economist, Vilfredo Pareto made the famous observation that twenty percent of the population owned eighty percent of the property in Italy. In 1941, management consultant Joseph M. Juran came across this work and realized it was a general tendency of many kinds of processes for 20% of cause to produce 80% of effects. This has often been observed in software development. Much of the project can be designed and coded in a relatively short time, but then much more effort is often needed to test and debug the code and get it to pass the acceptance tests.

This implies two types of effort. If productivity is defined as the ratio between effort and result then the initial effort is 16 times more productive than the later phase of the project,

(0.8/0.2 = 4 whereas 0.2/0.8 = 0.25, therefore 4.0/0.25 = 16) Can the emphasis of the effort be changed so that the latter part of the project is as productive as this first part?

The principles described in the first section of this document can be applied to this problem as follows: The central activity of the first stage of the project is creating an automated expression of knowledge within the problem domain. The knowledge-centic focus of the effort is extended throughout the remainder of the project by creating an automated expression of knowledge in the solution domain. This is done primarily by extensive use of unit testing. Several other techniques are used as well. They are all interconnected within the Eclipse IDE (Integrated Development Environment). Use of this automation will generally cut a project down to at least half of the best-case amount of effort required using manual test and validation.

We also make use of design patterns in our projects. Although these patterns are generally implemented in C, they are equivalent to an object-oriented approach. Object oriented programming is often described as having three basic characteristics: data abstraction, inheritance, and late binding. The knowledge structure in many problem domains is often hiearchical. A inheritance hierarchy can be created that mirrors this structure. In C++ terms, if virtual functions are used, the knowledge structure is decoupled from execution of functionality. The knowledge structure is stored in a data structure, the vtable and functions are called by reference. This can result in a much more efficent project that can also be very flexible. However, it is often not so hard to implement an equivalent arrangement in C.

We have several sample projects. Click the “contact us” link on the home page and we can arrange for you to access them through the clients' login.