Agile Before Agile Was Cool

Gordon Sleve
Robbins Gioia LLC

Success can be achieved by many means. Sometimes it is obvious which road to take, other times it does not really matter. Look at individual circumstances before choosing one path over the other.

People go years, possibly their entire lives, exhibiting certain behaviors, sometimes knowingly but often unaware of any notable pattern. Sometimes an event will occur where a name is given to their particular behavior. A man who takes his work problems out on his family may be in an anger management class and be informed that he exhibits *misplaced aggression*. For a woman whose husband is an alcoholic, yet buys liquor for him, might be labeled an *enabler*.

This identification can lead to an epiphany for the subject. This sudden realization is exactly what happened to me when a colleague of mine inquired as to my willingness to write this article on agile programming.

I had never before heard the term agile programming, but my associate was familiar with the concept and also with my work, so I was willing to trust in his judgment. During research into the concept of agile programming, it quickly became evident that this was an acceptable label for the coding practices I have used routinely for more than 13 years.

Unwitting Agile Programmer

In my work as an analyst for a program management firm, I use a fourth generation language (4GL), control and analysis tool to build reports and graphs. My customers and I use these documents to perform analysis on schedule related data. The information derived from this data is used to point out past mistakes and potential problems, thus saving both time and money. This is my goal as a program management analyst, and the goal of my firm, to make our *customers successful*.

Occasionally I am called upon to develop related applications or modules using 4GL. Some applications are written to analyze existing data and some to capture new data. The latest major development effort involved a resource-forecasting tool used to project future work requirements, and provide *what-if* scenario modeling. Because the customer wanted to keep the existing analysts at the site working on their current assignments, a separate contract was written and programmers were hired to perform the work. This project followed a traditional software development methodology because this methodology worked for this project. The project finished on time and under budget.

In 1994, the Air Force was awarded the Navy FA-18 programmed depot maintenance (PDM) contract. This was historic in that it was the first time that one branch of the armed forces was contracted to repair a weapon system used by a different branch. The accepted proposal called for the work to be performed at Hill Air Force Base (AFB) in Ogden, Utah.

Since the fall of 1993, I had been working as a program analyst on the Programmed Depot Maintenance Support System contract in the Aircraft Directorate at Hill AFB. When the new workload arrived, the program management team provided precedence network schedules and tracked the work against the plan, as had been done for the other aircraft work at Hill AFB.

Not long thereafter, the Aircraft Directorate was audited by an independent audit agency. Their findings required Hill AFB to provide an auditable *unplanned work* approval tracking system. The F-18s were brought to Hill AFB for a PDM, which is basically an overhaul of the entire aircraft, according to a specific set of operations to be performed, called planned operations. There are also provisions for problems encountered either during aircraft inspection or while the mechanics are performing planned work. These problems are defined as unplanned work and require extra time not accounted for in the planned operation package.

Given that the FA-18 is a Navy weapon system and each aircraft has spent a good deal of time on aircraft carriers or at coastal Naval Air Stations, much of the unplanned work is corrosion removal. This unplanned work accounted for more than half of the total hours on a FA-18 PDM. The independent audit agency required a way to show that hours billed to this contract workload were not being used to work on F-16s or C-130s, which were located in the same hangars as the FA-18. Without such an auditable system, the workload would be pulled from Hill AFB and given back to the Navy.

The Aircraft Directorate quickly established a manual process that may have satisfied the requirements set forth by the independent audit agency. The only worry was, with hand carrying of thousands of documents to and from the hangars, some were bound to get lost and therefore the system might fail the audit.

An Air Force major working in the Aircraft Directorate approached our firm about automating this process. It was easy to show a good potential return on investment (ROI) based on the amount of man-hours involved in processing work cards in the manual system. This also fell into the scope of our firm's program management charter. Automating this system would provide the ability to add the hours generated by this unplanned work into the schedule as soon as the requests were approved, thereby extending the schedule in a real time fashion. Since all parties were in agreement that this was a *win-win* situation, the next decision was how would we proceed with development?

To bring in more analysts would require writing a new contract or making an amendment to the existing one. Since the customer wanted the system in place by the time the audit agency returned, this option would take too long. They decided instead to reallocate the existing resources of the program management team and place all three analysts on full-time development of the new application. Due to the time constraint, there was no development of a formal plan or extensive requirements, which led to the use of methods now described as agile.

A Perfect Agile Fit

Traditional programming methodologies were put in place mainly to prevent *requirements creep*. In agile programming, potential for these problems is diminished by getting the application to the users quickly. Surely if it takes two years to develop an application, changes will arise in the organization or process that will drive new requirements and cause delays. Agile methodologies exist that are specifically tailored to long-

term projects, but my experience has been with short-term projects. The unplanned work module was not extensive and we were confident that we could provide a quick turnaround. Even though we did not have agile methodology guidelines to go by, this project was a perfect candidate for just such a philosophy. The Agile Software Development Manifesto states:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
> - Individuals and interactions over processes and tools.
> - Working software over comprehensive documentation.
> - Customer collaboration over contract negotiation.
> - Responding to change over following a plan.
>
> That is, while there is value in the items on the right, we value the items on the left more. [1]

Although we did not know of agile methodologies, in retrospect we practically followed them to the letter. We focused almost entirely on those *items on the left* and for all intents and purposes, ignored the *items on the right*. In this case, circumstance rather than conscious thought led us to perform in this manner. We were under the gun to get a working product in place in a short amount of time. There was little time for planning, contract negotiation, or documentation.

The lack of planning shows a slight departure from agile, but remember, we did not have the agile methodology with which to work. The available resources and their expertise locked in our tools. Due to the fact that very little of the traditional methodologies were available to us, we were forced to draw heavily from what is now an agile approach. I like to look at these two approaches as the Scarecrow and the Tin Man, agile being the Scarecrow (flexible) and traditional being the Tin Man (rigid.) They both want to get Dorothy to Oz, but they each have unique abilities and weaknesses. In our case, we had to rely on a Scarecrow named "agile."

The initial requirements were to automate the manual process and add reports. The bulk of the detailed requirements were obtained during development by working closely with civilian counterparts involved in the unplanned work process. These individuals were planners, schedulers, and the approval authority, each with knowledge of how their piece of the process worked.

We met daily, but on an informal basis, usually with the point of contact (POC) most familiar with the section being worked in a one-on-one setting. This would almost always take place at the developer's desk. We showed progress from the previous day and verified with the POC that their requirements were being met. Then we would identify any changes that needed to be made and start working to that end. As we moved through the development phase, we would identify those *nice to have* features and record those requirements for follow-on work. Through this process the programmer learned much more about his customer's needs than he could by reading thousands of pages of requirements documents. It also showed a commitment to individuals and interactions over processes and tools.

As changes were made to the application, corroboration was sought from the user before continuing further. Sometimes the user would sit through several iterations of code changes right at the programmer's desk, commenting and critiquing. By working closely with the user community, there was very little need for documentation and training. By the time development was completed, the users had been trained through their constant involvement. There was no need for a formal training class after implementation. Since we knew there would be a good deal of follow-on changes, we decided to wait on documentation.

The main application was completed with great success in plenty of time for the next audit. The only cost associated with the program was the temporary loss of productivity toward the original program management objectives. Our team was awarded a certificate of appreciation from the Aircraft Directorate citing a savings of 250 direct/indirect man-hours per aircraft. There was a dollar figure placed on both the cost (approximately $28,800) and the savings (approximately $1 million) resulting in an actual ROI of more than 30 to 1.

Given the limited planning work, there was no way to predict such a fantastic windfall. However, had more time been spent in planning, documentation, contracting, and processes the ROI ratio would have been less impressive. Follow-on changes included expansion for both F-16 and C-130 workloads, and a *master write-up* module that provided the users with a pick list of frequently used write-ups.

Specified Successful Use Continues

Although the F-18 contract only lasted one year before the workload was returned to the Navy, the use of this application on the F-16 and C-130 programs continues to this day. This program is still in a textual interface format but will soon be converted to an Oracle/Web-based format to provide better accessibility and ease of use. The user community has taken ownership of this application and they like it because of that very fact; it is their own creation. From time to time, minor bugs appear, which will be the case when omitting configuration management and strict coding practices, but the users are happy with that trade-off for flexibility.

One might argue that agile software development *flies in the face* of program management. While the irony of a program management professional touting "responding to change over following a plan" is not lost on me, I see both methodologies coexisting and filling an important purpose: to *make our customers successful.*

Yes, I am a proponent of agile programming but only in those cases where it is the best solution. Unless given a specific set of circumstances, I cannot say which is best. I have experienced success with both agile and traditional software development methodologies, and success is the ultimate goal.◆

References

1. AgileAlliance. "Agile Software Development Manifesto." 13 Feb. 2001 <www.agilemanifesto.org>.

About the Author

Gordon Sleve is a senior program analyst for Robbins Gioia LLC working in the ICBM program office at Hill Air Force Base (AFB) in Ogden, Utah. He was previously a site manager at Letterkenny Army Depot in Chambersberg, Penn. Sleve was selected as Robbins Gioia's Senior Program Analyst of the Year for Dayton-based operations in 1995 for his support of the implementation of Programmed Depot Maintenance Support System at Hill AFB.

Robbins Gioia LLC
OO-ALC/LMSO
6014 Dogwood Ave.
Bldg. 1258 Rm. 14
Hill AFB, UT 84056-5816
Phone: (801) 775-5943
Fax: (801) 586-4835
E-mail: gordon.sleve@hill.af.mil