**Spring**

2012

ARINC 838 Project Reflection Report

Team Information Overload V2

Mike Deats, Scott Griffin, Ryan Neal, Brandon Sutherlin, and Liron Yahdav

08

**Fall**

Contents

[Did we Succeed or Fail? 3](#_Toc323845232)

[Team Dynamics 4](#_Toc323845233)

[Burnout 4](#_Toc323845234)

[Metrics 5](#_Toc323845235)

[Process 5](#_Toc323845236)

[Conclusion 6](#_Toc323845237)

# Did we Succeed or Fail?

The short answer: We succeeded with flying colors. However, the devil is in the details, as they say, so to fully understand that claim we have to dig bit deeper, starting with how we measured our success.

Any project that has even the slightest chance of success needs a set of goals to strive toward. We started this project with three goals:

* Identify shortcomings in the ARINC 838 Draft Specification, and help to mature the standard.
* Simplify the loadable software installation process
* Evaluate whether Agile methodologies can be utilized to complete projects in the aerospace industry, particularly where certification using DO-178B standards are required.

Obviously, goals alone are not enough; we needed some way to determine if we had successfully met these goals. Thus, we proposed the following success criteria:

* **Goal #1:**
  + Document any shortcomings in the standard and identify a minimal set of data sufficient to load software successfully
* **Goal #2:**
  + Implement a basic software loader and verifier using the ARINC 838 standard
  + Create a proof of concept module that can handle installation of software to varied avionics systems
* **Goal #3:**
  + Document our experience (either positive or negative) in using Agile

Of course with any software product, the real measure of success is how happy is the Customer with our product. Moreover, being an Agile team, we worked closely with our Customer throughout the process to ensure we were meeting his expectations. While we did not create a fully functioning “proof-of-concept” module, we did complete the basic software loader and verifier, which was our Customer’s primary concern.

In addition, we kept track of all issues and shortcomings we identified in the specification and made sure we discussed them with our Customer on a regular basis. This was so effective that we actually received a second draft of the specification early on in the project that incorporated many of our recommendations. We feel this shows that our analysis has been effective at identifying gaps within the specification.

Finally, we delivered an analysis on our experience with Agile as a viable methodology for aerospace products. This was one of the most critical concerns for our Customer, and while we are a long way from creating a “certifiable” product, we found Agile to be an extremely effective process for creating software that has the rigor required by the aerospace industry.

# Team Dynamics

It is not a secret that our team specifically requested to work together. Our experience in the Foundations course was so phenomenal and successful, that we wanted to recreate that same experience during our capstone project. The biggest challenge we faced was not that we did not get along, or some members put in more work than others did; our biggest challenge was simply synchronizing the schedules of five people with full-time jobs, families, and other distractions, spread across two time zones and three states. To that end, we found that scheduling regular working sessions at the same time every night was an extremely effective way to ensure we were all on task and focused, even if not everyone could make it every night. We also lived and died by our shared Google Calendar, which was an invaluable tool at making sure everyone knew exactly what we had planned.

To put it simply, for teams that are completely or partially remote the most important factor in keeping everyone motivated and focused was to meet regularly as a team before splitting up to code or work on documentation. This is even more important during the early stages of the project as it takes time for a team to find its rhythm.

Another effective tool we learned in Foundations that we put to good use for this project was the use of dedicated roles, and roles rotation. Each week, one person was assigned the role of Facilitator (responsible for keeping everyone focused, and handling disagreements), and one person was the Scribe (responsible for keeping track of meeting minutes and other notes). At the end of the week, we rotated roles such that no one had a role for two weeks in a row.

The only problem we encountered was dealing with an odd number of teammates. Much of the work was done in pairs, following the Extreme Programming paradigm, but this always left one person as the “odd man out.” We handled this by either assigning work that was fairly straightforward and simple to the single team member, or by tackling a particularly difficult problem as a triplet. This approach worked fairly well, although we never found a smooth way of handling the odd number; it simply had to be addressed at each work session.

## Burnout

As the end of the semester neared, we began to suffer from a team-wide burnout. There were several reasons for this. Many team members were in the process of searching for new jobs and two of us ended up starting new jobs during the project (one person moved to a new city as a result). Several other members had major job-related events that resulted in late night or even weekend work, resulting in other members having to pick up the slack. Another problem was that our team was spread across different time zones, so in order to have everyone be available, our working sessions always started late in the evening.

There really is not a way to control real-life events, and the reality of this project is that at its core it was an academic project. As such, it had to fit within both our personal and professional lives, which made it even harder to strike a healthy work-life balance when pursuing this degree program as a part-time student. In most cases, we had to sacrifice our personal time to devote to our academics, and when both our academic and professional lives demand more of our time; our personal lives are the ones that suffered.

# Metrics

At the beginning of the project, we decided to track the following metrics:

* Code coverage
* Story points
* Average velocity
* Effort logged (total and per story)
* Time spent coding

Most of these metrics we got for free though our tools, e.g. code coverage via our TeamCity continuous integration server, velocity via Pivotal Tracker, etc. These metrics proved valuable as it showed us where our testing was weakest, and gave us a good indication of just how much work could be done. Unfortunately, the effort-based metrics had to be tracked by hand, as none of our tools had the capability to track that kind of information. We created a spreadsheet to track our time each day and early on in the project, we did this quite diligently. However, as the project progressed several team members stopped updating the spreadsheet, and by the end we had simply stopped tracking those metrics altogether.

There were several reasons why we dropped the effort metrics, but the primary reason is that we simply stopped seeing any value in those metrics. We felt our velocity was good, stories were being completed, and above all: Our customer was happy. Nevertheless, even with that in mind, we might have continued to track that data if our tools easily allowed us to enter time against stories or tasks. In the end, we realized that in order for a metric to be useful it has to have value and be as painless as possible to collect. In addition, we should have made an active decision as a team to re-evaluate our metrics partway through the project rather than simply ignoring them. This would have been an excellent mid-project exercise.

# Process

As is the case with many Agile projects, we tailored the process (in this case Extreme Programming) to fit our goals and our team’s situation. Being completely remote, we did not have the luxury of being co-located, or setting up work areas that were conducive to the concept of informative workspace. However, we did our best to emulate this type of environment using many powerful Web apps such as Pivotal Tracker, Google Docs, and GitHub. Since our project was released as open source under the MIT License, we were able to procure the use of a free TeamCity Continuous Integration server provided by CodeBetter.com. All these tools were invaluable in keeping our team informed and organized.

The other tenants of XP, iterative development, user stories, whole team, test-driven development, and even pair programming (through screen-sharing technology) worked well for us as they don’t tend to suffer from the team being remote. We did deviate somewhat when we did our pair programming. As previously mentioned, having an odd number of developers made it difficult to divide into pairs. Either we tended to work one team as a triplet, or had someone work solo on a trivial problem.

One of the most valuable tools in the XP toolbox is the use of reflection meetings at the end of each iteration. This is the time that team and process problems are identified and adjustments are made to overcome them. In short, process is there to prevent failure and not to guarantee success. Unfortunately, we became victims of our own success in that we felt we were doing so well we began skipping reflections at the end of the last few iterations. Had we done these reflections as we had planned, we might have identified earlier that we were beginning to suffer from burnout and were losing momentum.

Another Agile concept that we utilized was YAGNI or “You Aren’t Going to Need It,” which means to only implement the minimal design needed to accomplish the goals of the project. For most of the project, this worked well as we were able to spin up and begin coding extremely early. One consequence of this approach is that there is often significantly more refactoring. In traditional approaches, this results in schedule slippage or overtime, but for Agile this is a planned inevitability. We had one major refactoring event that resulted in the loss of one week of coding. However, had we taken the time to fully understand the requirements that drove the refactor it would more than likely have taken more than a week to plan and design.

# Conclusion

Team IO2 set out on the ARINC 838 project not only to develop a reference implementation of the ARINC 838 specification but also to explore the use of Agile development methodologies in the aerospace industry. We feel we have accomplished the primary goals of the project, and have delivered significant value to our customer. We fully implemented the specification in software, identified gaps in the draft specification, and provided valuable data to our customer on the use of Agile development methods in aerospace. The philosophy behind Agile software development is “embrace change” and even in a project as short as ours that concept served us well and allowed us to leave a fully satisfied customer, and a product to which our team is proud to have contributed.