**Spring**

2012

ARINC 838 Engineering Notebook

Team Information Overload V2

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

08

**Fall**

Contents

[Engineering Notebook 3](#_Toc323845104)

[Welcome to the ARINC 838 wiki! 3](#_Toc323845105)

[About 3](#_Toc323845106)

[License 3](#_Toc323845107)

[Overview 3](#_Toc323845108)

[What is the ARINC 838 project? 3](#_Toc323845109)

[Conventions 3](#_Toc323845110)

[Branch naming convention 3](#_Toc323845111)

[Decisions 3](#_Toc323845112)

[Decided to implement the XDF file first 4](#_Toc323845113)

[Ditching Interfaces 4](#_Toc323845114)

[Verification and Validation 4](#_Toc323845115)

[Reading and Writing Binary Files 4](#_Toc323845116)

[Strings in BDF will not escape special XML characters 4](#_Toc323845117)

[CRC Implementation 4](#_Toc323845118)

[Validation of input files 4](#_Toc323845119)

[Big Refactor 4](#_Toc323845120)

[Path for Files 5](#_Toc323845121)

[Interesting Discussions 5](#_Toc323845122)

[Risk Profile and Impact 5](#_Toc323845123)

[Agile Releases 5](#_Toc323845124)

[Eclipse Setup Tips 5](#_Toc323845125)

[Setting up the IDE with Maven and Git 5](#_Toc323845126)

[Importing code templates and style templates 5](#_Toc323845127)

[Installing the TestNG Plugin 6](#_Toc323845128)

[Installing TeamCity plugin to Eclipse 6](#_Toc323845129)

[Logging into TeamCity from Eclipse 6](#_Toc323845130)

[Process 6](#_Toc323845131)

[Iteration Length 6](#_Toc323845132)

[Release Cycle 6](#_Toc323845133)

[Effort estimation 7](#_Toc323845134)

[Acceptance Criteria 7](#_Toc323845135)

[Roles Rotation 7](#_Toc323845136)

[Process Reflection 7](#_Toc323845137)

[Iteration 0 (23 Jan 2012 - 5 Feb 2012) 7](#_Toc323845138)

[Iteration 1 (6 Feb 2012 - 19 Feb 2012) 7](#_Toc323845139)

[Iteration 2 (20 Feb 2012 - 4 Mar 2012) 8](#_Toc323845140)

[Spring Break (5 Mar 2012 - 18 Mar 2012) 8](#_Toc323845141)

[Iteration 3 (19 Mar 2012 - 1 Apr 2012) 8](#_Toc323845142)

[Iteration 4 (2 Apr 2012 - 15 Apr 2012) 9](#_Toc323845143)

[Iteration 5 (16 Apr 2012 - 23 Apr 2012) 9](#_Toc323845144)

[Releases 9](#_Toc323845145)

[Release 0.1 9](#_Toc323845146)

[Release 0.2 9](#_Toc323845147)

[Risks 9](#_Toc323845148)

[Initial Risks and Mitigations 9](#_Toc323845149)

[New Risks 10](#_Toc323845150)

[Third Party Tools 10](#_Toc323845151)

[Issues 11](#_Toc323845152)

[Tools 11](#_Toc323845153)

[Language 11](#_Toc323845154)

[Libraries 11](#_Toc323845155)

[IDE 11](#_Toc323845156)

[VCS 11](#_Toc323845157)

[License 11](#_Toc323845158)

[Continuous Integration 11](#_Toc323845159)

[Issue/task tracker 12](#_Toc323845160)

[Planning and tracking - Pivotal Tracker 12](#_Toc323845161)

[Bugs/tickets: Github Issues 12](#_Toc323845162)

[Builder 12](#_Toc323845163)

[Unit test 12](#_Toc323845164)

[Documentation 12](#_Toc323845165)

[Collaboration Tools 12](#_Toc323845166)

[Hex Editor 12](#_Toc323845167)

# Engineering Notebook

This is a snapshot of our [project wiki](https://github.com/squirrely/arinc_838/wiki) as it appeared on April 27th, 2012. This is provided only for archival purposes, This document will not be updated or maintained in any way. The project wiki is the source of truth.

# Welcome to the ARINC 838 wiki!

## About

A prototype system capable of installing various types of software onto a mock avionics unit following the ARINC 838 standard. Watch the progress of the project with [Pivotal Tracker](https://www.pivotaltracker.com/projects/457281/overview).

Built by [CodeBetter](http://www.codebetter.com/)'s [TeamCity continuous integration server](http://www.jetbrains.com/teamcity). See the current status [here](http://teamcity.codebetter.com/project.html?projectId=project179).

## License

All software is provided under the [MIT Software License](https://github.com/squirrely/arinc_838/blob/master/LICENSE)

# Overview

## What is the ARINC 838 project?

ARINC 838 is an open source, prototype software loader that is designed for use in the aerospace industry. It follows the ARINC 838 draft specification for defining software components to be loaded on avionics line replaceable units (LRUs). This project is an attempt to validate and implement the specification.

# Conventions

## Branch naming convention

In Git we should create a branch for each feature, chore, bug, etc. Liron suggested a good naming convention:

* feature-12345-brief description
* chore-12345-brief description
* etc.

Where 12345 is the Pivotal Story ID.

# Decisions

Information about decisions made that do not affect tools, process, etc.

## Decided to implement the XDF file first

Most of the team is more comfortable with XML than binary. It was decided to implement the XDF story first in order to learn the spec in a more comfortable standard. Confirmed that this was acceptable with Chris 1/25/2012.

## Ditching Interfaces

We decided to scrap the interfaces. Initially we had decided to use interfaces to define the spec in code. We realized that this was only adding maintenance overhead without adding any value. We may add the interfaces later, but for now we are removing them.

## Verification and Validation

Each setter will be responsible for validating data as it is set. The setter is responsible for validating everything that is in the scope of the field. For example, that the value is a UINT32, is in the accepted range, but will not check if the field is correct with respect to another field. Validation that requires multiple fields will be done by the first object that is the parent of both fields.

## Reading and Writing Binary Files

Given the requirement to read and write binary files in a random access manner, we decided to use the [RandomAccessFile](http://docs.oracle.com/javase/6/docs/api/java/io/RandomAccessFile.html) class in Java. One caveat is that this file reads and performs all operations in memory. We believe that these files will be relatively small, so we are not concerned with this. However, if performance issues arise, this may be a culprit.

## Strings in BDF will not escape special XML characters

Special XML string characters '<', '>', and '&' stored in the BDF will not be escaped as is done in the XDF file.

## CRC Implementation

Examined the java CRC 16 and 32 bit implementations described in the following links. 16 and 32-bit CCITT CRCs (<http://introcs.cs.princeton.edu/java/51data/CRC16CCITT.java.html>) (<http://introcs.cs.princeton.edu/java/51data/CRC32.java.html>) The java implementations were fast and similar to the custom implementation provided in the ARINC 838 Spec, however the resulting integers did not match. The team decided to port the custom crc from C to java

## Validation of input files

If the XML or binary is not well formed, the program will indicate that the file cannot be read. No further validation or verification will be attempted. Once the input files are determined to be well formed, further verification will be done to indicate that the files match the spec.

## Big Refactor

We encountered an issue with the verification story. It caused a large refactor of the application. Our mid-semester [presentation](https://docs.google.com/a/west.cmu.edu/present/edit?id=0AatJVf7DbhNKZGRxYzdqd3hfMTI0ZndtZ2Q3aHA) centered around this issue.

## Path for Files

The spec is not explicit regarding the path of the files that are defined within the spec. We created a gap bug for this for clarification. We believe that the path issue is either not relevant given aerospace architecture, or so common an issue that the spec writers did not clarify. We are going to add a path parameter to our SoftwareFileDefinitionDao object to alleviate the issue.

# Interesting Discussions

## Risk Profile and Impact

Is it a high impact with a low probability or a low impact with a high probability?

## Agile Releases

Had a good discussion on 1/30 about how to map our milestones to a schedule. We did not want to apply hard dates to the milestones (or releases), but the statement of work asks for them. We discussed how to handle this in the most Agile manner. The consensus was to apply dates, but stipulate that they are complete guesses. Then clarify how the estimates will improve as we work. Talked also about the importance of delivering value each iteration, though the customer may not have enough “value” to use at the end of each iteration. This allows the customer to decide if the project is worth continuing.

# Eclipse Setup Tips

## Setting up the IDE with Maven and Git

1. Open Eclipse, create a new project, Maven project, and select “Checkout Maven project from SCM”
2. Select ‘git’ as the SCM URL. If Git is not available, click the link “Find more Maven SCM connectors in the m2e Marketplace” and install the ‘eGit’ plugin. Restart Eclipse when prompted, then repeat steps 1-2
3. Add ‘git@github.com:squirrely/arinc\_838.git’ for the URL then click Finish
4. Once the project has imported, right-click on the project, then goto Team -> Share project
5. Select Git as the repository type
6. Check the “Use or create repository in parent folder” and be sure the “arinc838” project is selected.

## Importing code templates and style templates

1. Open Eclipse properties (Windows: Window -> Options, OS X: Eclipse -> Preferences)
2. Expand Java -> Code Style
3. Select “Code Templates” and then import the ‘misc/eclipse/arinc838\_code\_templates.xml’ file
4. Select “Formatter” and then import the ‘misc/eclipse/arinc838\_code\_formatter.xml’ file

## Installing the TestNG Plugin

1. Open the Eclipse Marketplace: Help -> Eclipse Marketplace
2. Search for TestNG. You should find the "TestNG Plugin" for Eclipse.
3. Install the plugin. Restart Eclipse.
4. You should now be able to run the TestNG tests from Eclipse.

## Installing TeamCity plugin to Eclipse

1. Open “Help” -> “Install New Software”
2. Click the “Add” button at the top of the window to add a new software repository
3. Type a name (e.g. “TeamCity, provided by CodeBetter”) and paste this link into the location field:<http://teamcity.codebetter.com/update/eclipse>
4. Click “OK”
5. Check the box next to “JetBrains TeamCity Integration for Eclipse
6. Click “Next”
7. Agree to the terms and conditions, then click Finish.
8. Restart Eclipse when prompted.

## Logging into TeamCity from Eclipse

1. Open “TeamCity” -> Login
2. Paste the URL [http://teamcity.codebetter.com](http://teamcity.codebetter.com/) into the field, and enter your TeamCtiy username and password. Check the “Remember me” box so you don’t have to ligin each time Eclipse starts

# Process

XP, with Pair Programming required only on critical pieces of code

## Iteration Length

2 weeks

## Release Cycle

Two releases of approximately 6 weeks in length (three iterations)

## Effort estimation

Planning poker with Story Points. We use [PlanningPoker.com](http://planningpoker.com/) for estimation sessions

We decided to estimate on a Fibonacci scale (1,2,3,5,8) to allow for a wide range of estimates, and to fit within the limits of Pivotal Tracker.

## Acceptance Criteria

Coding on a story should not be started until Chris has been notified that acceptance criteria have been written and we are awaiting his approval.

## Roles Rotation

Roles are rotated every week, after the meeting with Professor Katz. Roles rotate down along the list below, and wrap from the bottom.

* F = Facilitator - Responsible for manging meetings and ensuring discussions stay on track. They are also responsible for settling disputes or conflicts that arise during meeting or working sessions.
* S = Scribe - Responsible for recording meeting minutes or notes, and generating documents that will be used to capture thoughts and ideas during working sessions. This is not an absolute role as anyone can write notes during meetings. This is beneficial to the scribe, who often misses the chance for discussion or interaction due to attempting to capture the discussion points.

Role Rotation List (Week of 4/16)

* Brandon
* Liron - F
* Mike
* Ryan - S
* Scott

# Process Reflection

Team IO 2.0 is using Extreme Programming (XP) to develop the Arinc 838 project. This is unusual for software development in the aerospace industry, which tends to prefer traditional software development methodologies. This document captures our thoughts about the process and any changes we make to the process through each iteration.

## Iteration 0 (23 Jan 2012 - 5 Feb 2012)

This iteration was mostly infrastructure and environment setup, so no development was done.

## Iteration 1 (6 Feb 2012 - 19 Feb 2012)

In Iteration 1, we anticipated that we would be able to split the work more. We believed we could do boiler plate code and tests on an individual basis, rather than in pairs, to be more efficient. This assumption didn’t pan out. We ended up having to work together to conquer the learning curve associated with the spec. We had identified our lack of understanding as a risk and we saw that manifest. Our strategies of relying on the customer and our use of pairing helped to mitigate the consequences, but still our productivity was impacted. As a team we still met our goals for the iteration, but we had originally planned to be able to do more of the work in parallel.

Since the team also works full time, most of our meetings were at night, and went late into the evening. This contributed to burnout on each member, directly impacting productivity. The team didn't see large-scale impacts from this, but we were seeking to find ways to segment the work such that more could be done individually. As the design solidified, our understanding increased, and our productivity increased when moving from larger pairing sessions (5 people) to smaller ones (2-3 people).

Our reflection session of this iteration identified that we should verify acceptance criteria of all stories with our customer before we start implementing stories, to ensure we implement the desired functionality.

## Iteration 2 (20 Feb 2012 - 4 Mar 2012)

As this was our second coding iteration, the team was more in a flow, and although most of the work was still done in pairs, there was better segmenting of the work. We managed on several occasions to be very efficient with segmenting the work to the point where we had two pairs and one individual working all in parallel. This is probably the most efficient segmenting we expect to have on the project. The only potential for further efficiency gains would be to identify additional individual work that could be done by people when it is not possible for pairs to meet up.

## Spring Break (5 Mar 2012 - 18 Mar 2012)

## Iteration 3 (19 Mar 2012 - 1 Apr 2012)

We hit our first major obstacle this iteration. We learned that the verification code needed to be completely separated from all other code. This is due to DO-179B requirements. While we had a story and a partial understanding of this requirement, the depth of it was ignored by the team until our customer explained it fully. We conscientiously made this decision, though we probably did not realize the required depth of the separation. This has caused us approximately a week's worth of refactoring work. It also caused us to miss our goals for this iteration. We only delivered one of the anticipated three stories. So did Agile fail us?

We feel that Agile not only did not fail us, but has served us well, even taking into consideration this obstacle. We did have to do some refactoring, but at a cost of one week, it was relatively minor. We likely could have eliminated this refactoring with some additional planning at the start of the project. However, how much planning? There is no way of knowing how much planning and knowledge we would have had to get under out belts in order to understand this requirement at the level that we do today after a 30 minute discussion with our customer on it. The same discussion at the start of the project probably would have required much more time, had we even known the question to ask.

While we did lose some time, we feel that we gained more by diving in and delivering value to the customer. A traditional planning phase could easily have lasted a month or more. Meaning that we would barely now be delivering meaningful code to the customer, instead of being deep into the second phase of the project. The customer was accepting of the cost of the refactor, and is happy with the value received. Having two months worth of working code makes a week slippage much easier to accept.

## Iteration 4 (2 Apr 2012 - 15 Apr 2012)

Our team is really in a groove at this point. We haven't encountered any obstacles and are just trying to finish strong. Mid-iteration we determined that we needed to identify what was most important for our customer. It was likely that we would not have time to complete another major section. Therefore, we discussed the highest priority features with Chris mid-iteration. He indicated that his major goals had been met, so we could focus on whatever we deemed interesting. We decided to finish up the CRC stories and "tie the bow" on what we had accomplished so far, attempting to finish the verification section.

## Iteration 5 (16 Apr 2012 - 23 Apr 2012)

We cut our final iteration short as we needed to focus on our client and school deliverables in the final week and a half of the semester. However we were able to finish all our verification stories, which was a major milestone. We also had a real reflection meeting as a team, which we hadn’t done for the last few iterations. During this meeting, we discussed the issues we had with the CRC stories where we ended up having to do a rather painful code merge. This was because we simply didn’t coordinate well enough.

We also reflected on our metrics choices. Originally we planned on tracking our effort, both total and per story, but it became too tedious and we stopped tracking. We found that if the tools don’t easily allow us to track our metrics, we won’t track them. We should have actively decided, as a team, to drop the metrics rather than doing it passively by simply stopping.

There were good things that came out of our reflection meeting. We were very good at self organizing, which meant we spent more time working and less time organizing as a team. We also did a pretty good job dealing with the odd number of team members, balancing the more complex tasks with the pairs or triplets.

# Releases

## Release 0.1

First release on March 3rd, 2012. Download [here](https://github.com/downloads/squirrely/arinc_838/PP-ARINC-838-LSP-Release_0.1.zip). sha1: **ef771be4a782a79c3f73e7543cacae0d885a4816**

## Release 0.2

Last release on May 4th, 2012. Download [here](https://github.com/downloads/squirrely/arinc_838/PP-ARINC-838-LSP-Release_0.2.zip). sha1: **f77fa26795c87cab9a72befe5e2e0812cf0c9afc**

# Risks

## Initial Risks and Mitigations

1. Difficulty understanding the specification
   1. Rely on Client’s domain knowledge
   2. Utilize pair programming to disseminate knowledge
2. Real jobs and life events take precedence
   1. Keep calendar up to date and team informed
   2. Anticipate individual work and personal life loads and plan accordingly
3. No experience with loadable software tools
   1. Rely on Client’s domain knowledge
   2. Utilize pair programming to disseminate knowledge
4. More features and school artifacts than the schedule and available resources permit
   1. Divide work between pairs to balance artifacts and code
5. DO-178B constraints
   1. Utilize pair programming to keep high coding standard
   2. Use SCM system to track commits to stories
6. The standard is still early in its life-cycle
   1. Utilize Client’s close contact to fix developing problems
7. Gaps in tool knowledge, license knowledge, etc.
   1. Utilize school resources to understand constraints
   2. Continually be aware of license
   3. Utilize pair programming to maximize domain knowledge

## New Risks

### Third Party Tools

A new risk was identified, unfortunately not until it was an issue. Currently the project relies on several third party tools, GitHub, Pivotal Tracker, TeamCity, etc. Prior to the first release we experienced outages in several of the tools, fortunately none during a critical time. However, on the weekend of the first release, we had an outage that did affect our ability to deliver a non-critical document in the timeframe we desired. While this was not a critical issue, its occurrence prompted us to identify this as a risk.

### Issues

We have discussed our risks at the end of each iteration. While we have seen life events, the spec's newness, and our lack of experience affect the project, nothing has been beyond the scope of our mitigation plans.

# Tools

## Language

Java 6 (specifically 1.6\_14 or higher). We considered 1.7, however it’s newness and lack of full cross-platform support (e.g. not yet available on Mac OS X) led us to chose Java 1.6. We also considered using .NET, however team familiarity was highest with Java.

## Libraries

* [JAXB](http://jaxb.java.net/) - For XML support
* [TestNG](http://testng.org/doc/index.html) - For testing
* [Mockito](http://code.google.com/p/mockito/) - For unit testing

## IDE

[Eclipse Indigo](http://eclipse.org/indigo/) - Selected due to team familiarity. Free. Netbeans was considered, only long enough to be rejected, though we may revisit this for UI work. JetBrain’s IntelliJ was also considered. However, the free version is limited. We have collected several [tips](https://github.com/squirrely/arinc_838/wiki/EclipseSetup) for setting up Eclipse to work in our environment.

## VCS

Github - Selected due to team familiarity with Git. Successful use in previous projects. No other options were seriously considered.

## License

[MIT](http://en.wikipedia.org/wiki/MIT_License) - Selected due to its brevity. Also because it is one of the most liberal licenses. With the target audience being the aerospace industry, we wanted to make the code as accessible as possible.

## Continuous Integration

[TeamCity](http://www.jetbrains.com/teamcity/) (via [Codebetter.com](http://codebetter.com/))

* Scott: sgriffinusa
* Mike: SQUIDwarrior
* Liron: lyahdav
* Ryan: lone.squirrely
* Brandon: brandons

[Project website](http://teamcity.codebetter.com/project.html?projectId=project179) We chose TeamCity because several of our members have direct experience with the tool and have found it to be far superior in usability and features to other options such as Hudson or CruiseControl. There is also a free TeamCity server hosted by Codebetter.com that is available for OSS projects. We also considered Heroku, but that is more targeted to hosting websites.

## Issue/task tracker

### Planning and tracking - Pivotal Tracker

Pivotal is one of the best online tools for planning and tracking Agile projects. All team members are familiar with Pivotal through our CMU coursework or in our jobs. It is also free to use for OSS projects. [Project site](https://www.pivotaltracker.com/projects/457281)

### Bugs/tickets: Github Issues

Because our project is hosted on Github, utilizing the Github issue tracking system is a natural choice. One of our members has used this system before and had a good experience. We also considered Codebetter’s YouTrack server, but that does not integrate as closely with Github. [Project site](https://github.com/squirrely/arinc_838/issues)

## Builder

Team familiarity. Interest in learning Maven from those who have experience with Ant.

## Unit test

Initially we decided to use JUnit. However, Chris recommended looking at [TestNG](http://testng.org/doc/index.html). No one had experience with it, but since it is built on JUnit, the learning curve should be light. We decided to use TestNG to gain exposure to the tool.

In order to run tests easily from Eclipse, it is useful to install the TestNG plugin. Instructions are here: <http://testng.org/doc/download.html>

## Documentation

OpenOffice or MS Office (undecided at this point). Is there any restriction on using Microsoft office in an open source project?

## Collaboration Tools

Since we are a remote team, we need good collaboration tools to maintain our productivity. We decided to use Mikogo for screen sharing purposes, due to it’s excellent performance record. We also will be using Skype for both audio and video conferencing. We have used Adobe Connect in the past, but it tends to be unstable and the performance is often unacceptable.

## Hex Editor

Chris told us that Notepad++ has a hex editor plugin that is very useful when editing and creating binary files.