# Term Project Iteration 1 Walkthrough

Phillip Escandon
escandon@bu.edu

# Project Direction / Overview

Aerospace Camera System Error Logs, Maintenance Logs, Configuration Files-

During the process of building and integrating a camera system and all of its subparts, many log files are captured but not widely used. When called upon to review the logs, only a few team members have the background and historical knowledge to review and decipher any issues as well as suggest a course of action to rectify these issues.

My proposed database will contain log files and the associated describing data for each of these camera systems. I have collected approximately 9000 of these logs dating back to 2012. My goal would be a database that could quickly create a report(s) to:

1. Give a historical background and timeline of the camera, including SW versions, Configuration of the camera system when it was delivered to a customer and common failures observed during testing.
2. Give a historical background and normalized boot sequence for a given camera system.
3. Give a historical background of *failures* and some basic configuration and state that the camera was in during a failure.

Some of these items may seem trivial,but the collection of these logs and *quickly* deciphering them has always been an issue.

This database could potentially be used by four (4) teams.

The data to be captured and used in the database are the logs that are particular to each product.

Three particular logs will be used:

Errorlog.log
Maintenance.log
Sensor configuration

**Error Log**

In an abstract framework, the logs can be considered to consist of three sections:

- **Boot Section** - describes in terse, software terms, the boot sequence of the product.
- **Start Up Built In Test (SBIT)**- Once boot is completed, the camera synchronizes with a GPS clock and a built in test is conducted. It describes the status of the various subsystems in the product.
- **Plan-** describes the tasks allocated to the product and records the user interaction with the product until the product is shut down.

Hundreds of these logs are collected for each system. Each system has a distinct **SENSOR ID**. The logs are always called ‘errorlog.log’ or ‘maint.log’. Each errorlog will contain the Boot and SBIT section, but not necessarily the mission section. The maintenance log will contain Start times, end time, PlanID and SBIT Status.

Once the product is fielded for a customer and used, the Plan section will contain valid recorded data.

Current usage of these logs is minimal. Thousands of these logs exist, but they are rarely examined or mined for details and analysis. A shortened version of this log exists as a ‘maintenance log’ and is primarily used by the Field Service Team to quickly debug issues.

**Maintenance Log**

This is a simple version of the error log that only contains Plan ID, SBIT failures, startup and shutdown times. Everything in this log can be found in the Errorlog.

**Configuration File**

Each sensor has it’s very own configuration file. This contains values that were tuned during the integration phase as well as the final focus values for the various field of views.

# Interest

I have seen and used these logs but feel that they are being overlooked. Sloppy warehousing, incomplete datasets and timeframes, no real direction and no plan to use these logs or versions of these logs in upcoming projects.

For one particular vexing issue, I was given a small dataset of roughly 50 files- a list of keywords and general instruction as to see if I could figure out what was happening.

After some initial parsing and cleaning, I was still left with smaller and still very obtuse files. I could see what was happening, and could verbally explain but the team required context. They required a story to go along with this data. A parsing of the files did not provide the needed story and background.

During this parsing phase, I decided to focus on splitting up each log into three distinct sections, the boot section, SBIT section and mission section if it was available.

Previously - I have parsed these files using Python. I have subsequently moved some of the parsing to Windows Powershell, simply because it is ubiquitous with all Windows computers and I can use it when overseas working with a customer computer that might be limited in ability.

I now have a one pagefile with output that speaks to the relevant points found in the error log but am still missing some important details. The general questions that should be answered are:

- What Sensor had an issue?

Sensor ID

- When did it have an issue?
  Date
- What is the SW load for each component?
  SW Versioning information
- Did it pass SBIT?
  Test Report - state of the system before mission
- What was tasked in a mission?
  What was planned?
- What failed?
  What planned task did not work?
- What passed?
  What planned task did work?
- What keys were pressed on the system?
  What buttons did the user press during operation?

# Use Cases

**Integration Team**

The database could be used to identify issues with the product prior to release and shipment of the product. This team also captures data related to tests that are conducted in the 3 month build cycle, as well as a final configuration file.

This file contains tuned filter values for the controls system as well as focal and optical information. The database would provide historical data to be used as comparisons to the Unit Under Test and to ensure that the UUT is in family for certain attributes and values. (-+ 3 sigma from the mean.)

**Customer Engineering / Chief Engineers Group**

By keying and reflecting on a complete report, or series of reports generated by the database, a coherent story that describes successes and failures for individual systems can be generated. An engineer assigned to support country X, could quickly find all sensors delivered to X, SW versioning information and historical data. Decisions as to the exact SW load delivered and any required updates could be tracked and seen using this database. New log files could be inserted and reports generated to provide actionable direction to correct issues.

**Product Support / Quality Control**

Failures of the product could be captured in the database in the form of a test report generated from the SBIT section of the errorlog.log. The database could also be used as a final sell-off deliverable as proof that the product has indeed passed all required tests and document corrective actions that rectified failed tests. The serial numbers and of major components could be captured and used by the Production and Quality team.

# Fields

| Field | What is Stores | Why it's needed |
|---|---|---|
| Sensor ID | Stores the ID of the individual camera | Each camera system is unique and has this unique identifier. |
| TimeDate | Time and Date of the camera operation | This is the time and date field from the logs in Unix Epoch time. This will be converted to GMT time for the database. |
| PlanID | Identifies the specific planned use and configuration of the camera | This identifies the specific plan that was used during the operation. This is not a unique field. |
| SW Version | Software version of the four computer boards | This is the SW build version for each computer board. Usually does not change but is very helpful for historical reports. |
| Message | Message string from the logs | String message from the error log. This usually gives the details of pass / fail actions. |
| Pass/Fail | Boolean for testing purposes | This will give us a quick overview of the status of a test / retest sequence. |
| TimeSynch | The time that the GPS and computer times were synchronized | This gives an indication of system boots. If more than one Time Synchronization is seen, this could indicate a problem where the camera unintentionally restarted. |

## Fields (Continued)

| Field | What is stores | Why it's needed |
|---|---|---|
| RollResolver.Devicepub.offset | Offset for the Roll Resolver on camera | Historical Data - needed for comparison to other systems |
| RollResolverConfig.offset | Resolver configuration offset | Historical Data - needed for comparison to other systems |
| PitchResolver.Devicepub.offset | Offset for the Pitch Resolver on camera | Historical Data - needed for comparison to other systems |
| PitchResolverConfig.offset | Resolver configuration offset | Historical Data - needed for comparison to other systems |
| BS.nf1bm0 | Backscan Z transform filter value | These values are constantly looked up and reused. Historical Data - needed for comparison to other systems |
| BS.nf1bm1 | Backscan Z transform filter value | These values are constantly looked up and reused. Historical Data - needed for comparison to other systems |
| BS.nf1bm2 | Backscan Z transform filter value | These values are constantly looked up and reused. Historical Data - needed for comparison to other systems |
| BS.df1bm0 | Backscan Z transform filter value | These values are constantly looked up and reused. Historical Data - needed for comparison to other systems |
| BS.df1bm1 | Backscan Z transform filter value | These values are constantly looked up and reused. |

| | | |
|---|---|---|
| | | Historical Data - needed for comparison to other systems. |
| BS.df1bm2 | Backscan Z transform filter value | Historical Data - needed for comparison to other systems |

## Summary and Reflection

The database idea seems too big. Feels like this is too much to chew on. I feel scope creep as I think on the possibilities of fields and what I could do, so I'd rather focus on something that can be accomplished during this course and used to build upon after.

Concerns:

1. Should the error logs be preprocessed before going into the database? The sections of the log files are thought experiments and there are no demarcations in the logs themselves.
2. Should I parse and create the BOOT sections, SBIT sections and Plan section and use these to input into the database? Can I simply use the database to parse the needed fields?
3. The configuration section is a little ad-hoc. Each Camera Sensor has its particular Configuration File, dictated by the 'SensorID' field. The use case is primarily to find and flag possible errors… all configuration files contain resolver offsets between 20 - 28… but camera #6 has an offset of 40… why is this not in family?