1. Executive Summary (will be done at the end if all discussions are consumed)

Safety as a given thing by certification is out of scope.

Economic impact of software in relation to hardware is an increasing concern.

Data and facts based resolution process versa who cries loudest

Etc…

1. Introduction
   1. Purpose (I will still add this on the base of the APIM)

Ref to APIM, Phase 1, Phase 2

* 1. Audience ( I will add some text)

Suplier, Airframer, Airlines, RMO

* 1. Situation (I will summarize section 3 for this later)

Gen. Picture of SW development and how it came to aviation

* 1. Solutions (I will summarize as we go)

Make ref. to the appropriate chapter

* 1. Terminology (I will add later a few definitions)

SW failure, unexpected System behavior caused by SW etc….

* 1. Related Documents (help needed from IT – I can still lookup aviation)

IT industry, Aviation, ARINC Specs….

ITIL (IT-Infrastructure Library) is a widely used IT Service Management Framework

Official ITIL website: [www.axelos.com/best-practice-solutions/itil](http://www.axelos.com/best-practice-solutions/itil)

1. Defining the Problem Space (This section is almost complete but needs the review from all of us)

Software especially Loadable Software in aviation is controlled by a Part Number. Airplane Parts are supported in the field by the part manufacture/supplier and the airframe manufacturer. The level of support is defined in a commercial document that help the airline to get reasonable defined product support. Especially in a single source situation the Airframe Manufacture (airframer) agrees with the parts manufacturer a minimum or standard level of support that will be handed over to the airline as a support commitment from the parts manufacturer and a right to claim for the airline. (Mostly used term for such commitment is PSAA = Product Support Assurance Agreements). In the PSAAs parameters and metrics are defined that should be observed. In a case where those metrics cannot be achieved by a product the airline deserves the right of PSAA defined countermeasures to compensate for the underperformance.

Today the majority of metrics are based on MTBF and BTBUR. This is very hardware focused.

Today the majority of measures to compensate for underperformance are also hardware focused. Those are (but are not limited to):

* Additional Spare Parts supply FOC to offset the low stock situation
* Commitment for analyze of problem and development for a fix
* Onsite Support by a parts manufacture representative
* Mitigation methods, compensations and more

For software today this well working traditional Product Support Model does not work anymore to full satisfaction. Where software is not specifically excluded from support also no specific inclusion exists in most Product Support Agreements. As a result no specific measures are defined that would apply if an underperformance exists.

One main reason for this insufficiency is today’s absence of usable metrics to evaluate the functional and/or quality compliance level of software based functions. Consequently what cannot be measured cannot be judged.

The charter in the APIM 16-001 for Phase 1 was to evaluate possible parameters that would allow building metrics for software compliance as we have it for hardware today by MTBF and MTBUR.

Furthermore the group was chartered to explore possible ways how to technically acquire those key parameters and how to work with it.

The commercial part of definition for this group is per APIM out of focus. But it must be acknowledged that a total separation between the commercial impact and the technical definition cannot be maintained. Other AEEC/AMC Standards also define very detailed components of product support. One good example is the “ARINC 674 STANDARD FOR COST EFFECTIVE ACQUISITION FOR AIRCRAFT LIFECYCLE SUPPORT “

Without assessing today’s product support model for software in aviation and comparing this with other industries than the aviation industries product support models for software only a fraction of the problem space can be explored. Especially in the IT Industry pretty good guidance material exists and could possibly be used or partially adapted for the unique situation in aviation. At least it should be used for a benchmark analyze on what is missing in aviation that would exist anywhere else in IT, car, communication or other industries.

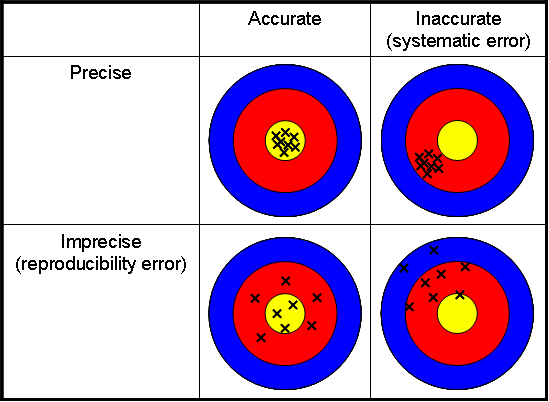
The endeavor to measure software performance as it really exists in the IT or communication industry today can be translated into an aviation desire to have something like Software Health, Trend or Performance Monitoring.

The root cause why we don’t have transparent commitments to the user for software products or software dependent functions in aviation is the lack of metrics to define any target.

The other problem is that a pure metrics for software cannot be judged without the functional impact of a system. For the end user it is not possible to differentiate a failure from a combination of soft- and hardware effects where it really belongs to. Only the functional effect is visible today. Also it could happen that one software part is not performing well but due to the fault tolerant design on a different instance of functionality the malfunction is not evident to the end user for years and the operation of the aircraft is not impaired. For some in our industry it appears not really desirable to monitor such situation.

A definition of the term “software failure” is difficult. Software does not have random errors and shows no wear and tear. If something is incorrect, the same failure shows up again and again if the environment is reproducible. (Unfortunately this is often not the case. All the crashes you see on typical PCs occur due to “random” combinations of external factors.)

On the other side, software can have systematic errors if requirements have been incorrect or incomplete, or the implementation is faulty.



In consequence the replacement of an Airplane Software Part with a new Software Part of the same P/N (and revision) can only result in continuity of a problem and is therefore not very advisable. Consequently the MTBUR statistic for a horrible part of software is often perfectly good. The MTBF is today only fractional recorded since known bugs are ignored for Technical Logbook Reporting (MaReps or PiReps).

Consequently someone could think it is too complicated and not providing sufficient clarity if we develop Software Quality Metrics. But this is not true. Other industries have developed methods how to assess software quality by parameters and how to use those in a product support environment (e.g. aggregated parameters like Service Level Agreement, Resolution Processes, or pure tech. parameters like Reset Rates, Start Up Times and Availability Measures, etc.…)

One other part in the problem space is the lag of standards and transparency between the stake holders. Especially in the division of software supplier, airframe manufacture and airline/MRO different level of understanding of existing software problems exist. First not all data in the problem databases of all players are shared between the effected members of our industry. Second there is no common format for such data. Also IP rights are hindering such data transparency. Anyway in other industries it was accomplished to develop open source formats and tools for this purpose. The same approach may be used in aviation. Some suggestions can be found in Section 9 of this report.



1. Research existing standards (this from the APIM implies non-aviation Std. I hope I find a IT person to help with this – Viktor, Martin?)

* ITIL,
* open source formats for logging system events
* open source tools to analyze logged information

1. Defining types of in-service problems (I started here but it needs more explanation)

We can define major categories of software related problems with Impact to the operation of an A/C or fleet of airplanes at the operator (airline).

1. Passenger Impact (priority 1)  
   Software issues impacting the comfort of the passenger or reducing the perceived value of the cabin product. (e.g. seat control software failing, no dimmed lights, air-conditioning control software failure)
2. Cargo Carriage Impact (priority 1)  
   Software issues impacting the capacity or category of cargo which can be carried. (e.g. cargo hold temperature control software, cargo steering control software failure)
3. A/P Turn Time (priority 2)  
   Software issues impacting the tasks to be performed during the turnover of an A/P including signing of logbook entries. (e.g. software malfunction preventing required information is provided, malfunctions observed during flight and documented in the logbook which can’t get easily resolved)
4. A/P Use Time (priority 2)  
   Software issues impacting the time the aircraft con be used for revenue service. (e.g. frequent complete aircraft power cycle required, software issue forcing operation to follow time-consuming workarounds)
5. Reduced Mission Capability (priority 2)  
   Software issues impacting the capability of performing flights as planned based on approved data. (e.g. SW issues impacting ETOPS capability)
6. Flight Planning (priority 2)  
   Software issues impacting the ability to plan the aircraft operation as usual. (e.g. SW issues impacting ETOPS capability)
7. Pilot Workload (priority 3)  
   Software issues impacting the workload for the cockpit crew. (e.g. SW malfunction triggering erroneous ECAS messages)
8. Cabin Crew Impact (priority 3)  
   Software issues impacting the cabin operation leading to more workload or reduced ability to keep level of service. (e.g. Software of Cabin Crew Attendant Panel freezes)
9. Line Maintenance Workload (priority 3)  
   Software issues impacting the routine work and leading to additional non-routine tasks to be performed. (e.g. Cycling the power of a system or an aircraft before flight)
10. Engineering Back Office Workload (priority 3)  
    Software issues causing additional workload within Engineering. (e.g. due to additional monitoring efforts, trouble shooting)
11. Training (all Pilot, Cabin, Technical) (priority 3)  
    Software issues requiring additional training to learn how it can be fixed (permanently or temporarily) or how to operate when the malfunction occurs. (e.g. loss of sensor information or erroneous data scenario)
12. Defining the criteria of classification of SW related problems

This point was taken from the APIM:

Should we do this or is section 5 sufficient? Possibly we need to define other differentiators. E.g.

1. Criticality of a function (safety) (out of scope)

2. Availability of a specified function (discussed in measurement techniques)

3. Operational Impact (in section 5)

4. Impact to economic aircraft operation (in section 5)

5. Possible effect of combinations of failures (don’t know what to do with this)

6. Impact to maintenance (in section 5)

1. Defining if additional information is required (I may add to this if we agree that data sharing is essential for transparency and success)
   1. Open sharing of data (Supplier <-> Airframer <-> Airline <-> MRO)

Several example from the Information Technology industry for open sharing of information on bugs exists. One can subdivide them by “Information-push” and “Information-pull”

* + 1. Information Pull  
       Primarily for addressing critical software bugs, companies in the IT sector invite white-hat hackers to penetrate their systems and put out rewards for found bugs. Usually those bugs are fixed by the companies before the bug is made public.
    2. Information Push  
       Software producing companies, especially those which bundle their software with a specific hardware (e.g. Apple) admit that their software is not guaranteed free of bugs. These companies provide in principle two alternative approaches to address this issue: user self-help and bug reporting.  
       In the category of user-self help means like forums or bulletin-board systems, but also twitter and facebook spport fall. Here users are able to share their problems with the user community and ask for assistance. These communication channels are also monitored by the respective companies.   
       Furthermore they also implement more and more means for automatic reporting of software bugs to their bug-tracking platforms.  
       Especially in the open source community those bug-trackers are made public in order to allow developers to coordinate themselves and to advise users on known problems.
    3. Common to both approaches is that the information on bugs is maintained with several relevant information in either public or restricted systems. Information inter alia includes *criticality*, *behavior*, *software revision*, *hardware revision* (when soft- and hardware are bundled), and *software-process* and *memory* information. In addition to this for identification of software related bugs in mobile devices also information on *position,* *temperature,* and other environmental information are stored.

7.2 Common formats of problem reports

7.3 Definition of minimum content of data fields of problem reports (just a minimum must be shared)

1. Propose measurement techniques

8.1. Human based measurements

Human based measurements are principal Pilot reports. Those are not very well calibrated in severity and frequency. Since the human judgment is included completeness and objectiveness is in question.

One concern is the pollution of the Technical Logbook with undesired squawks where a rectification by the certifying staff would be needed for each single entry in order to release the A/C for flight. This is what would need to be prevented.

The group discussed possible ways to enhance the pilot reports by objective parameters.

This could be done like a pilot triggered event report/snapshot (some ACMS system of some A/C maintain such a function). In such a situation the simple event is supplemented by relevant data and even a timeframe before and after the event can be recorded.

Note (Mark Christian):

Taking system snapshots would be ideal. We would need to come up with an idea about how this could be achieved. To cover also some time before the error was experienced by the user (e.g. an error in a less relevant systems starts small and sums up until it starts affecting other systems (e.g. memory leak)) some continuous recording might need required.

Also the trigger to take the snapshot would require some suggestions. Personally I’d prefer an automated trigger with an optional manual trigger as pilots would most probably first focus on maintain aircraft control and analyzed the situation before thinking of triggering a system snapshot.

Thinking about an automated trigger reverts back to the questions about what are the acceptable system parameters defined by the manufacturers/developers and how can these be observed during the flight?

One other way would be a specific function in an Electronic Logbook (eTLB) application. Especially if a portable EFB is hosting this application additional applications can be developed relatively easy (non-FAR 25 requirements – COTS based equipment).

So a report can be generated with relevant data being assembled without too much workload for the pilot. This could run in the general GUI of the EFB and with a minimum of selections by the pilot to gather meaningful information. The generated report would not interfere with the Tech Log function in regard to consider this as a rectifiable flight squawk.

To achieve a standard quality of human generated reports the principle of a decision matrix has been adopted to provide a framework to collect all relevant information in a structured way. The development of the matrix is in an early stage and feedback from the manufacturers and developers is required to identify the crucial parameters to limit the complexity and time required to complete the reports by the flight crew. Taking advantage of digitally available information to pre-select values should be considered for later implementations. The matrix should include all parameters and be suitable for all aircraft types while implementations for example into EFB applications to generate the report should only reflect the parameters which are relevant for the particular aircraft type.   
The final reports should be stored in an open, structured file format (e.g. XML) which allows processing and sharing the data without any pre-processing. All information which can be used to identify the airline or particular flight should be stored in a header section of the file, so it can be removed easily without influencing the remaining content of the report.

After the matrix has been finalized it could be used to develop the database model to store the information from the reports in a centralized database. This would allow the users to identify the occurrence of same or similar issues and their impact much quicker and potentially support the trouble shooting process.

8.2. Onboard machine based measurements

8.2.1 Self-Monitoring by the software function (Application)

Most software applications today maintain a self-monitoring or analytic function that provides the results in a storage area that can be interrogated by different means. This can be:

* a pure debug mode (only shop visit access)
* writing log files (can be interrogated by different means)
* reporting to a central maintenance device, (with or without programmable downlink function)
* a next higher unit, (central controller of a very federated system e.g. cargo loading)
* a post flight report, (generated by the CMC/CMF)
* a connected device (Broadband, ACARS, EFB, IFE etc..)
* etc…

Some types of software do not have practical access today under normal circumstances. But most software apps do have self-monitoring and any type of access for the user.

The access may be different for each software function and the format may also be different. Here is an area for a (ARINC) common standard for the future. See section 9.

Note:

Check ARINC 852 (“Guidance for Security Event Logging in an IP Environment”) for potential parallelism (e.g. format, parameters, methods)

Each software application should provide information on the following parameters:

Software Vital Signs (SVS) per function

1. Count of reset (per Interval)
2. Startup time (each case)
3. Read write inabilities - memory access issues (each case)
4. Out of memory events (each case)
5. No output data (each case)
6. No input data(each case)
7. Output / Input data out of range (each case)
8. Pointer stack overflow events
9. Free to use error codes ( 1-2) (each case)

Such definition of a minimum set of vital signs for a software application should be introduced to new system architectures. Such parameters still need more definition.

Not all systems must necessarily provide all ten parameters.

Self-monitoring systems must be connected to a central entity like a CMC/CMF or an onboard network architektur (ONS). This central entity should function as a report consolidator and off board router.

Legacy airplanes that obviously don’t have such architecture today may also generate data of this kind but the access is not very direct.

For those types of existing airplanes we already receive a lot of data on the ground. With the help of data conversion we can possibly do on the ground the same or at least a part of it compared to the new generation aircraft. See 8.3 for more.

8.3. Ground based measurements

Legacy airplanes but even more later models generate a good amount of data. The data sources are:

1. FODA
2. ACMS
3. Engine Trend Data
4. Data link traffic
5. Etc…

Those data (big data) can be analyzed by tools to understand inconsistencies and indications of software insufficiencies.

1. Field of opportunities for ARINC Standards
   1. Format Definitions of SVS (Software Vital Signs)
      1. As basis of a common format for transferring data from monitored avionics to ground applications for analysis, monitoring and prediction, an existing open format used in general IT is preferable.

The common format allows the usage of existing toolchains to receive, store and analyze the data vendor-independent. Further, the formats used in IT are prepared to be extended for the specific purpose to accommodate transfer for aviation and avionics specific type of data.

Two commonly used protocols are available that should be taken into account based on the data that has to be transferred:

“Carbon” is a basic protocol to transfer simple metrics in realtime and offers small aggregation possibilities to collect data. It is restricted to one metric per dataset though which limits the transmission of multiple measurements at once.

The protocol for the open source database “InfluxDB” is a (backwards compatible to Carbon) version that allows transmission of multiple measurements per dataset including additional tags. This protocol suits the usage for avionics monitoring very well since it allows multiple tags to narrow down the system that is sending the measurement (aircraft tailsign, system, version etc.).

Common for both protocols is that each measurement contains a timestamp so that they could be stored in databases that are especially designed for quick access to time series of data.

In case avionics do not provide means to aggregate and transfer data in common protocol formats, the transfer of log files is another (proprietary) option to at least gather the data from these systems as well. The decoding and transformation of the data needs to be done on the ground to possible convert it to the common formats and feed it into a common database.

The common protocol as defined by an ARINC standard should be a derivate from one of the aforementioned IT industry standards.

References:

Carbon: <http://graphite.readthedocs.io/en/latest/feeding-carbon.html>

InfluxDB:<https://docs.influxdata.com/influxdb/v1.3/write_protocols/line_protocol_tutorial/>

* + 1. Additional sources

In order to get more data for analysis existing data streams from avionics to ground (e.g. Datalink traffic including ACMS/CMCS reports, FODA, Engine Trend data) may be used. The ground systems receiving this data should be able to convert it into the common format as per 9.1.1 and feed it to the tools that collect it from different sources.

The transformation of these proprietary formats to the common protocol is usually depending on the data availability and existing tools used (e.g. MRO tooling, Datalink ground applications etc.). If needed, an ETL tooling could be used to get proprietary data, transform it and forward it to the data storage in the common protocol.

* + 1. Analysis tooling

When complying to a commonly used protocol, existing tooling may be used to store and analyze the data.

If ARINC proposes the usage of a specific tool, it should be one of the existing or at least a derivate of the existing tools as follows.

“Graphite” is a basic open source web application to graphically display data gathered in a time series database.

“Grafana” is the more universal standalone web application that connects to different datasources (e.g. InfluxDB) and allows for customized dashboards and analysis of data in different graphical and textual representations, even with display of geolocations on a map if available in the data.

References:

Graphite: <http://graphite.readthedocs.io/en/latest/tools.html>

Grafana: <https://grafana.com/>

Grafana Demo: <http://play.grafana.org/>

* 1. Transparent Problem Database

Transparent data sharing between all aviation participants (software supplier, airframer, airlines, MRO) utilizing a dataset framework defined to record all relevant parameters of software related problems to enable all stakeholders to identify similar issues and receiving all required information in a common format. This could be a subset of the data fields in the Problem Report DB that all airframer maintain. maintain (cf. examples for the Information Technology industry – section7.0).

Such database is mandatory for a TC holder of A/C that use software which has been developed in accordance with RTCA DO 178B. Those DBs are typically very extensive but for the purpose of a transparent data sharing device the number of required fields can be kept small.

The DB can be supplemented by data for the software supplier and the airline/MRO.

Beside the critical part of defining a minimal set of data fields, the hosting by a neutral, independent and safe organization is another challenge which needs to be resolved.

Here SAE / AEEC could be an option but we know of IATA initiatives that still have to be explored.

Nima:

For hardware related issues it is already best practice to collect data from airlines to derive actionable information (e.g. Boeing ISDP). First platforms which enable the user (AVIATAR) or the airframer (e.g. Airbus RTHM, Boeing AHM) to run state of the art data analytics algorithms on data also emerge.

Taking the example for the Information Technology industry it is safe to say that one key element for successful tracking of bugs and improving of software is the sharing of bug-related information with a community of developers and/or developers and users. In addition, since avionics usually process data measured by sensors, they are as environment-related as hardware elements.  
Therefore, as for consumer electronics, it is important to also store environmental, i.e. flight path-related, data together with the bug reports.

We deduce from this the need for a transparent, but access-restricted, neutral data platform which would enable for data collection. The stored data should remain within the possession of the data producer, i.e. the airline, with privileged access to this data through an independent application (or a number of applications) which would provide the airframer, the component OEM and the airline with actionable information.

* 1. Support Model

As done in ARINC 674 “STANDARD FOR COST EFFECTIVE ACQUISITION FOR AIRCRAFT LIFECYCLE SUPPORT” the AEEC or the AMC or a joint committee could work on a “STANDARD FOR COST EFFECTIVE SOFTWARE LIFECYCLE SUPPORT”

In order to get to a full picture of eventually existing insufficiencies in the product support model in aviation a benchmark with ITIL is suggested for phase 2.

ITIL is the most widely accepted approach to IT service management in the world. ITIL can help individuals and organizations use IT to realize business change, transformation and growth.

In the IT industry the ITIL best practice model is widely used for IT service management. A joined team of IT and Aviation experts should compare all well-established ITIL methods with the special requirements of aviation reality.

In the IT industry, the key metrics of SW (or even system) quality is put into Service Level Agreements as measured in the running system. Failures or even complete outages of an IT system are accepted for a previously agreed (and contracted) time-window. The whole assembly of ITIL processes is implemented in a way, to assure these SLAs in real operation. Typical downtime values for a system are between zero down-time and up to 4 hour reaction time. In addition to these operational topics, a wide range of other topics can be handled as e.g. SW release changes, maintenance changes or even questions by the individual end user of the system.

A success factor of this kind of IT system operation is of course the world wide connectivity via the internet. A not uncommon scenario is the location in different countries or even continents for call center, computer center and operations.

With the advent of increasing connectivity for the aircraft, this kind of methodology of system operation is increasingly progressing into the aviation domain. An example is the operation of IFE systems in an online/offline IT operation scenario.

* 1. Logfiles

In the area of the Onboard Network Systems (but not limited to it) of e-enabled airplanes log files are used to understand system events. Most of those log files are captured for security events. Also system performance and failure records are found. It is expected that those information are a growing source of information that can be used for software metrics. Unfortunately most log files are of non-standard formats (all different) but usually in an undocumented but easy to read ASCI format.

Here is a big opportunity for ARINC Standard making. (need to include a status of AEEC activity here)

1. Discussed Use Cases (here we may include Use Cases as we discussed it – possibly a summary and conclusions)
2. Related presentations (here we may add key presentations as we have seen it in the course of our meetings)
3. Open issues / clarifications

Not all Items could be worked in consensus in the Phase 1 WG. Here is a summary of the major disconnects. (this section is still quick and dirty - needs help from the 3 groups)

Software suppliers:

The software suppliers in general expressed that software metrics is difficult to establish and for the energy spend it would provide too little benefits. This was also the storyline of the presented Use Cases by the suppliers. Over and above all software suppliers believe we have good existing processes to warrant sufficient software quality. Also the fact that aviation software must be certified in accordance to required standards (Do 178B, DALs and more) for being approved to go on an airplane should warrant sufficient quality.

Airframer:

Only Airbus and Boeing participated in the Phase 1 WG. Both acknowledge the value of software metrics but also believe that processes in place are sufficient today and better methods of support on the base of metrics are not needed.

Airlines/RMO:

Airline participation was 5 in the 1st F2F meeting and was 2 in the second.

The airline participants where very motivated to archive changes or enhancements in monitoring software parameters, since the current situation especially in the domain of software product support is felt increasingly insufficient. Software problems are the biggest single growing aspect in A/C maintainability today. With ever growing use of software driving functions over pure hardware parts airlines believe we have to enhance the product support for software. This can only work with measurable parameters here software metrics.

The understanding of the suppliers and the airframers that everything is good without measuring success is not accepted.