Proceedings of the 19th Symposium

Aircraft Integrated Monitoring Systems

Garmisch- Partenkirchen, Germany

AIMS98

Aircraft Integrated Monitoring Systems

Published by: Arbeitskreis Telemetrie e.V.

The Importance of Testing for Successful Life Usage Monitoring Systems

Jürgen Broede, Hugo Pfoertner, Klaus Richter

MTU Motoren- und Turbinen-Union München GmbH
P.O. Box 50 06 40
80 976 Munich, Germany

ABSTRACT

Life usage monitoring systems contain two areas of fundamental interest for validation, namely the correct functionality of the life usage monitoring algorithms and robust implementation.

The functionality of the algorithms is validated against the entire lifing process. Since it is not possible to check the results of the monitoring algorithms against a statistically representative large number of missions calculated by the design tool, a numerical simulation of this process is required. A procedure based on the Monte Carlo technique has been developed, which allows the assessment of the deviation of the monitoring results in terms of average and scatter from the 'truth'.

To validate the correct function in the target system, recorded input parameters in form of digitised sets are fed in. With identical inputs to the reference programs and the on-board versions of the algorithms, it is possible to obtain bitwise identical results between both implementations.

Integration tests and flight tests concentrate on the robustness of interfaces and on the behaviour of the monitoring system under non-standard operating modes.

Accessibility and data transfer are important in early stages of development, e.g. with prototype installation. To involve intended customers of the monitoring system into the testing process often reveals problems not covered in the specification of system or software.

INTRODUCTION

When speaking about aero engine life usage monitoring systems, we understand systems which acquire aircraft and engine signals over whole missions, and process them into life consumption figures for selected critical areas. Life consumption is then accumulated over the entire engine life time. The life consumption data are tagged with aircraft and component part numbers and serial numbers, and are collected in a data base for the whole fleet. The fleetwide data in this data base allow for statistical analyses of life consumption, trend estimation, maintenance planning and spare parts management.

Generally, such a system is designed as a distributed system, what means that parts of the system are located on-board in the aircraft and other parts on ground at the

air bases and in logistic support centres. All the parts of this distributed system need to communicate with each other.

Details of the functionality and the architecture of aero engine life usage monitoring systems as well as experience gained over years in service were the subject of several presentations [1 - 8]. But one aspect very important for the success of such a system has not been much pronounced in these publications, namely testing and validation as part of system development and establishment.

REQUIREMENTS

An overall requirement for a distributed life usage monitoring system is that it works to the satisfaction of the customer. This general requirement determines the quality of the whole system and can be decomposed into a number of detailed requirements which specify (and also quantify) the quality characteristics of the system. Certain aspects of the system development process - particularly the quality management and verification and validation activities - are influenced as well.

Software quality characteristics are defined in ISO/IEC 9126 [9]. They encompass functionality, reliability, usability, efficiency, maintainability and portability. The effort necessary to achieve and proof these characteristics depends mainly on the criticality of the software and the system wherein the software is embedded. The criticality level of a system is normally contractually fixed.

The criticality of an item expresses the significance given to a functional failure of that item. The criticality is categorised in levels, where higher levels are related to more severe consequences expected for the case of a functional failure. Consequences considered are risk to people, economic losses or damage to property and environmental damage. Malfunctions of systems of the highest risk class may cause the death of many people and may lead to the ruin of large companies, whereas breakdown of an uncritical system (low risk class) may only put some inconvenience to the operating personnel.

Examples for attaching evaluation techniques to software characteristics and criticality levels are given in [10]. To evaluate the functionality or functional correctness of an item of the lowest level, simple black box testing may be sufficient, whereas the proof of functionality of an item of the highest level requires inspections, walkthroughs, traceability evaluations, black box testing, white box testing and formal verification. The reliability of a software system with low criticality level might be sufficiently guaranteed by using the facilities provided by the chosen programming language and by a users survey. The reliability evaluation of a critical system additionally calls for field experience, fault tolerance analysis, stochastic system analysis, a reliability growth model and as a final step also for a formal verification.

Quality cannot be achieved by only testing and assessing the final product. This means that testing is only one aspect of the quality assurance process. Quality assurance involves all kinds of software product evaluations. Checks should be applied for completeness, consistency, feasability, testabilty, coverage of all requirements, accuracy and adherence to plans.

Quality assurance measures for software related products typically include constructive measures (e.g. using software development standards, supporting the

development process by methods and tools). Development tools are available to support system and software specifications, architectural and detailed design. For example, excellent guidelines are available for the Ada95 language [11].

All formal measures to ensure software quality assume that system requirements and software specification are correct and complete. But the reality looks different. Experience shows that the operational environment, the interfaces to the operators and sometimes the handling of the data outside the on-board equipment are very vaguely described. Those deficiencies normally show up very late in the system integration phase. And also operational conditions never expected in the system definition phase turn out to be the normal daily practice.

Other specification deficiencies concern requirements not being testable. Generally, any specific requirement should be such that an objective and feasible test can be designed which is able to determine whether the requirement has been met.

For the success of a monitoring system it is also essential that the quality characteristics are not only required for the monitoring software itself but in a similar way for data bases, manuals, documentation and training material.

STEPS OF SYSTEM DEVELOPMENT

The first step in the development of a life usage monitoring system is (or should be) the definition of the overall functionality and the allocation of individual tasks to the different parts of the distributed system. In this phase often the accuracy requirements are specified as well.

In the following steps the algorithms and interfaces between the parts of the system are defined. Algorithm development itself is a process with a number of sub-tasks. The algorithms serve for the calculation of life consumption from the measured input signals.

The core tasks in the life consumption calculation process are the calculation of

- performance parameters
- transient temperature distribution within the components
- transient total stresses at the monitored critical areas
- the resulting life consumption at these areas

These core tasks are supported by additional tasks as

- detection of engine start (start criterion)
- detection of engine shut down (end criterion)
- input data check and correction
- input data filtering and synchronisation
- result check and correction

and by service tasks such as

- generation of data sets, containing engine configuration and life usage data
- data base management
- analysis of the fleetwide life consumption data with respect to trends, maintenance planning and spare parts management

Interfaces provide the relationship between two or more entities (which may be software items, hardware items or human operators) where data are shared, provided or exchanged.

Figure 1 gives an impression of some of the interfaces whose correct function needs to be proven during development and system testing. Whereas the interfaces in the upper part of the figure are accessible by usual test methods (simulation, debugging, comparison with independent reference results) most of the interfaces in the lower part of the figure involve man machine interactions, transfer of engine hardware together with information not stored in computers. Large differences in the training level of personnel and also in the organisational structure of the interacting parties add a great deal to the complexity of the data handling process. The only viable solution to guarantee long term integrity of the lifing data of the engines and components of a large fleet is to perform continuous checks on the accumulated information in the central logistic support system. Those checks include comparisons against statistical models derived from the analysis of recorded flight data as well as tracing single parts for continuity and plausibility. Detected errors need to be corrected before they could lead to an inadvertent usage of parts beyond their released life limits. Reasons for corruption of the data have to be analysed and corrective actions (adaptation of interfaces, training of personnel, shift of responsibilities) need to be proposed.

The next steps encompass formal software development, again separated into different phases, which are

- definition of functional requirements derived from the system specification
- software specification (including the algorithm functionality)
- software design and coding
- software testing (unit, component and integration testing)
- system validation

For formal software development a number of international (and national or project related) standards are set out which regulate the details of the process and the deliverables (e.g. [9 - 16]).

RTCA DO-178B [12]

RTCA is a private, not-for-profit organisation that addresses requirements and technical concepts for aviation. The products of RTCA are recommended standards and guidance documents that focus on the application of electronics technology to implement new or modified concepts and to satisfy related requirements. The DO-178B and its predecessor DO-178A form the basis for the certification of many on-board systems involving software, especially in non-military projects.

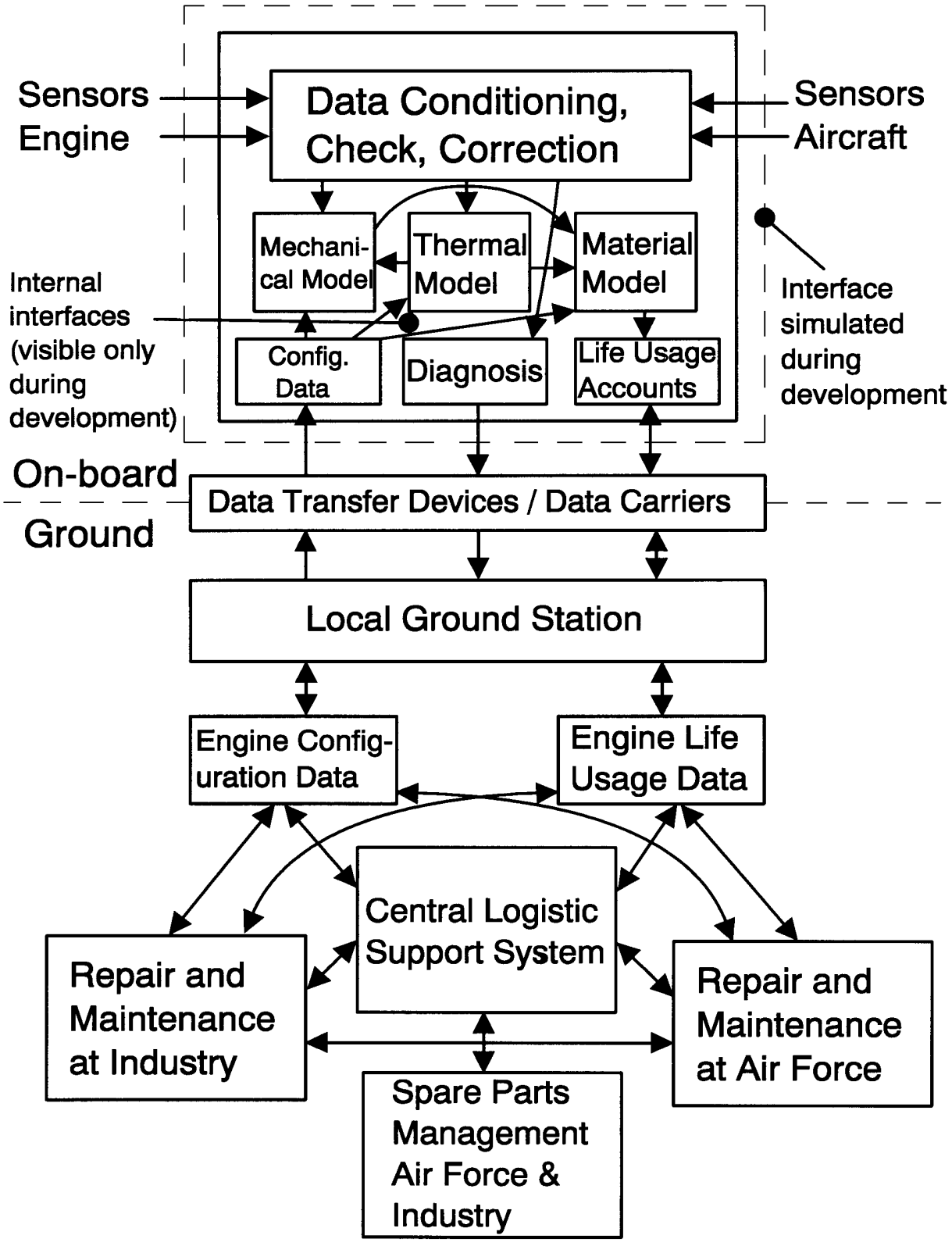


Figure 1: Interfaces of an Aero Engine Life Usage Monitoring System

DOD2167A [13]

The DOD2167A military standard (now superseded by [14]) has been used as the basis for the development of several engine control and monitoring systems throughout the world. The standard was designed especially for mission critical or weapon system software and puts its emphasis on the preparation of documents. This standard assumes a sequential "grand-design" model for system development, using a strictly top-down functional decomposition, which is usually not applicable to the incremental or evolutionary models being applied for the development of monitoring systems.

MIL-STD-498 [14]

The MIL-STD-498 standard was issued in 1994. It is far more flexible with respect to process models and also with respect to the documentation effort. Although we have not officially referenced this standard till now, parts of it (i.e. its associated DIDs [16] and the supporting reference material [15]) have been used as a very useful guideline for conducting development, testing and documentation in our actual projects.

Parallel to the software the necessary hardware is developed and validated, also according to detailed standards.

Very important in all these steps is the validation aspect. Two areas are of fundamental interest. These are the correct functionality of the life usage monitoring algorithms and their robust implementation.

ALGORITHM ACCURACY

‘Correct functionality’ of the monitoring algorithms should include a statement about the algorithm accuracy. The algorithm accuracy is a measure how far the life consumption quoted by the monitoring algorithms deviates from real life consumption.

And here already start the problems. The real life consumption of an aero engine part is not accessible. In fact, this would require to operate an engine part until failure. But nobody would do it, due to the hazardous consequences of such failure. Thus, only the predicted part life can be utilised.

Prediction of the life of aero engine critical parts is a sub-task of the design and engine certification process. Lifing activities start from predicted or measured engine performance data. Transient temperature distributions and stresses due to many types of loading are calculated. Stress and temperature histories for all potentially critical areas are considered in order to assess the ‘expected predicted safe life’ of a component. Tests are performed to verify the life predictions for the life limiting critical areas. Test evaluation - which includes effects of overload and scatter in life potential - together with the analytical prediction lead to the ‘predicted safe life’ of the component. The ‘predicted safe life’ of each critical engine part is formally declared in a Life Statement.

This lifing process in total produces the best information available, is accepted by the certification authorities, and can therefore be considered as a sound reference for development of life usage monitoring systems.

For the lifing process highly sophisticated tools (such as finite element codes) are employed. Intermediate results - which are performance data, temperature distributions, stresses and predicted lives - form a pool of basis data for the algorithm development. The basis data are representative for the whole range of engine operating conditions.

The life usage monitoring algorithms themselves are based on physical models and form approaches to the complex design models. The algorithms contain parameters which are optimised during the algorithm development process. The optimisation criteria are to minimise the deviations between basis data and algorithm results, both locally and globally.

Since the monitoring algorithms are based on physical models they allow for the interpolation between the basis data as well as for extrapolation to unusual engine operating conditions. In particular, it is evident that the algorithms basically behave as the complex physical model. Only quantitative deviations occur. The accuracy analysis is based on these deviations, which are statistically described as frequency distributions of the intermediate results. These frequency distributions are established as part of the algorithm development process for performance data, transient metal temperature distributions and critical area stresses.

The accuracy of the life usage monitoring algorithms is defined as the average deviation between the life consumption quoted by the monitoring algorithms and the life consumption predicted by the complex thermal and mechanical structure analysis used for engine design, when both are fed with identical input data profiles. Normally, the accuracy specification requires that the average of the quoted life consumption over several missions under real engine operating conditions does not exceed a specified window. Such a definition allows for larger deviations for rare flights with exceptional operating conditions which may not be accurately covered by the monitoring approach, but maintains the overall integrity of the monitoring system. A minimum value is defined to ensure that the quoted life consumption does not underestimate the real consumption in order to maintain flight safety requirements. On the other side, a maximum value is established which guaranties the quality of the algorithms.

This type of accuracy definition calls for a statistical validation procedure. The procedure chosen uses the Monte Carlo technique. The idea of the process can be outlined as follows.

The design procedure and the monitoring algorithms produce similar results for identical input data, since both are based on physical models and the monitoring algorithms have been optimised to closely match the basis data. The difference between the results is a function of the deviations which appear in the performance, temperature and stress modules. The deviations of the individual modules need to be represented as frequency distributions as sketched in Figure 2.

The difference between the results could be calculated if both the exact procedure and the monitoring algorithms were applied to a certain mission profile.

But it is also possible to calculate this difference directly from the deviations that occur in the different modules under this particular mission profile. For this route, it is necessary to identify the influence of these deviations on the final life consumption result. The calculations have only to be performed twice, namely once without deviations and secondly with the deviations which have to be superimposed to the intermediate results at the interfaces between the modules. Since both the exact method and the

algorithms behave similar, there will be no difference if one or the other procedure will be employed.

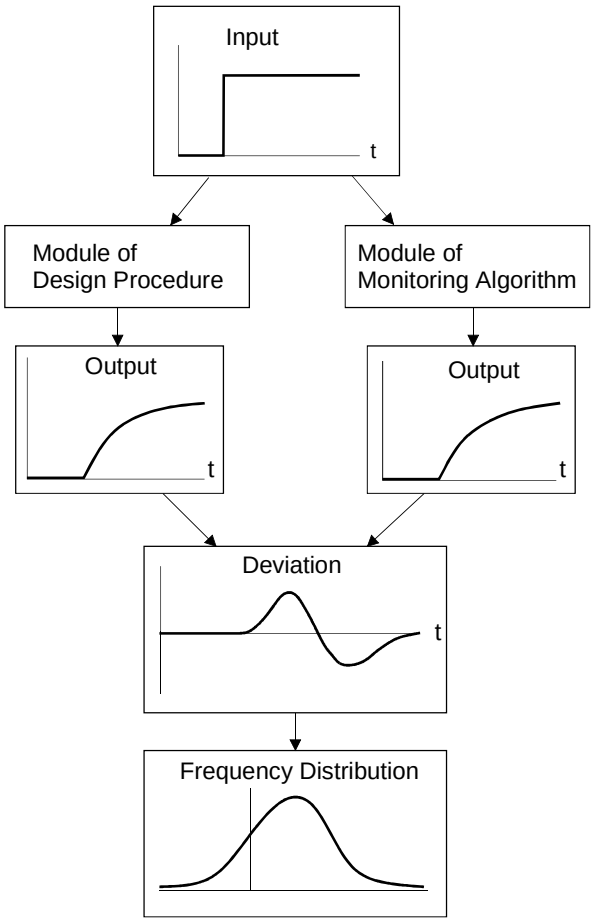


Figure 2: Deviation between Monitoring Algorithm and Design Procedure

The application of the exact design procedure needs large computing resources whilst the monitoring algorithms work very effectively. As the accuracy specification requires the consideration of average values of life consumption, not only one but a large number of calculations need to be carried out. For the statistical combination of the different deviation influences the Monte Carlo technique is used, which requires a statistically representative number of calculation repetitions. Therefore, it turns out that the only sensible way for validation of the monitoring algorithm accuracy is the second one where the influence of the deviations on the final result is identified by the application of the algorithms themselves. Figure 3 shows a flow chart of this process and allows for comparison between both routes. The route with dashed boxes is the one not chosen.

The analysis uses a number of representative mission profiles. In a first step, the missions are fed into the algorithm and the corresponding life consumption is calculated. These results serve as reference for the subsequent steps.

In the second step, the same mission profiles are applied but the intermediate results at the interfaces between the modules are modified with simulated deviations. These are derived from the deviation distributions which have been generated in the algorithm development process. The deviation actually applied is picked randomly from the relevant distribution curve.

The second step is repeated several times with randomly different simulated deviations, generating a variety of life usage results for each mission profile. These results differ from the reference result as well as from each other. For these results again a frequency distribution can be drawn and also average value and scatter calculated. The difference between the obtained average value and the reference result gives the quantitative measure for the accuracy of the monitoring algorithm.

VERIFICATION

Verification of the life usage monitoring system has the final aim to show evidence to the customer that the product meets all requirements. This includes also to assure

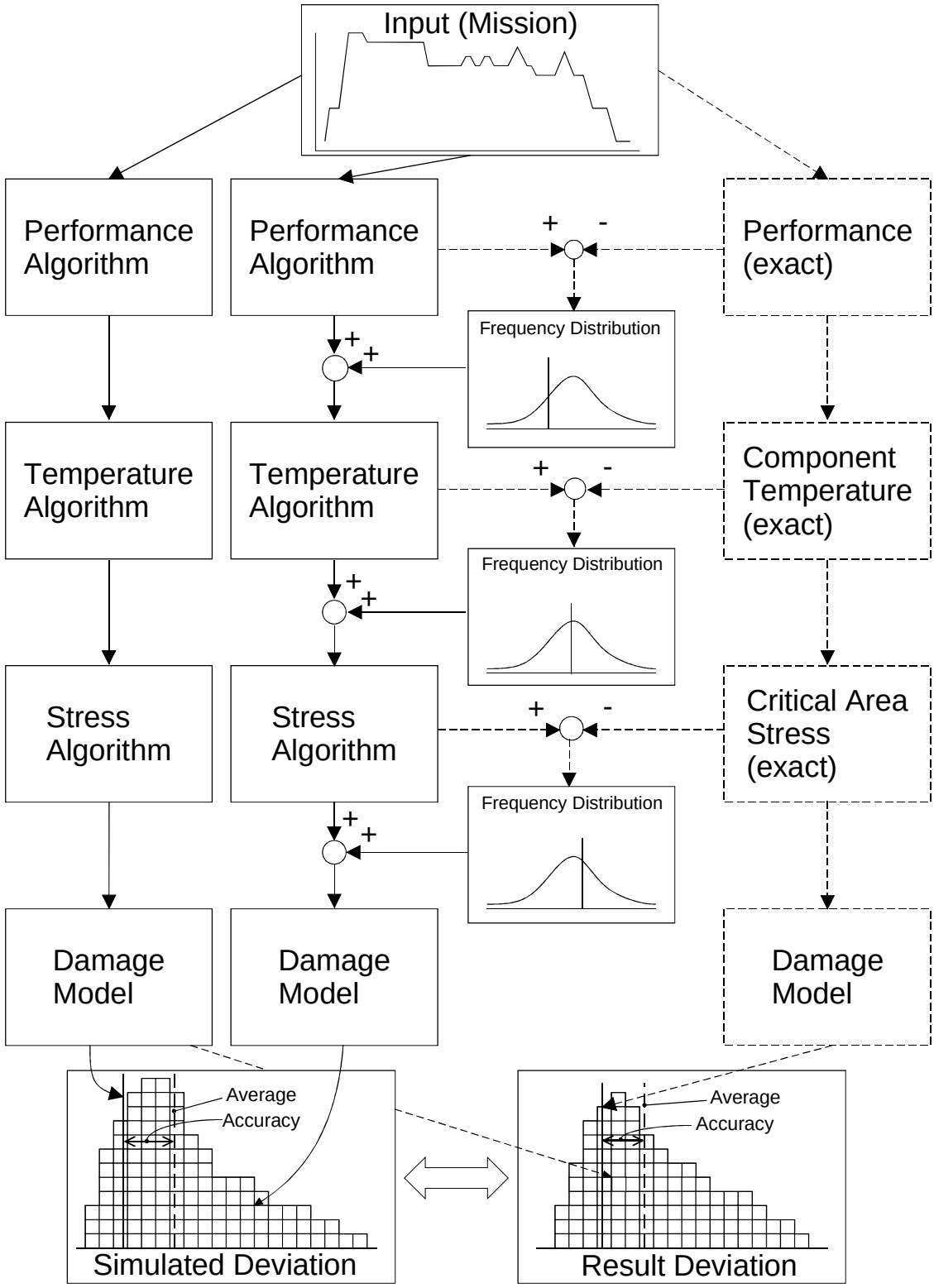


Figure 3: Accuracy Simulation

that the product does not contain other than the specified functionality. Tests failed at the first time need to be repeated after product modification until the tests are finally passed.

The whole verification activities are separated into a number of sub-processes which address the different aspects as correct algorithm implementation, software integration with the operating system and control software, hardware integration into the target computer system and eventually system integration of the complete monitoring system.

Software development standards distinguish between informal testing and formal testing. For formal testing, the principle of "independent verification and validation" is essential. This means that the persons responsible for qualification testing of a given computer software configuration item (CSCI) shall not be the persons who performed detailed design or implementation of that CSCI. This does not preclude persons who performed detailed design or implementation of the CSCI from contributing to the process, for example, by contributing test cases that rely on detailed knowledge of the CSCI's internal implementation.

Our own experience with several projects shows that it is not necessary to have independent personnel for testing already in early phases of development, where testing is considered to be only informal. The aim of informal testing is the identification, reproduction, localisation and elimination of bugs. A few highly qualified developers of software responsible for all aspects of development, including participation in system design and also internal testing, equipped with suitable tools may perform highly competitive not only in terms of productivity but also with respect to low error rates when compared to large development teams with a strict separation of responsibilities.

Where independence is really considered to be necessary we have made positive experience with the so-called multi version programming, where an independent person creates another instance of a CSCI based on the same specification, but without knowledge of the implementation details of the first version. This method has the advantage of providing a much better motivation to the tester giving him the chance not only to detect errors in someone else's work, but even to replace the original version with the new one, if this new one outmatches the item under test with respect to correctness or performance.

Many software development procedures require to start software testing with unit tests. We found that this may be not very practical as it needs a lot of work without any real benefit. Coding errors are much more effectively detected if software units are already integrated to functional groups and these software groups extensively tested, where the same result is achieved in shorter times with less effort.

Another point of concern is the re-use of software items. When dealing with similar projects - e.g. life usage monitoring software for different engine types - it is quite natural that some amount of functionality is the same. As software is designed in a modular way, there are a number of software modules which are identical in different projects. This allows for re-use of these modules. The concern is now the testing of these modules. Is it really necessary to test already validated software items again and again?

One of the simplest means to reduce the expense of software testing is to re-use

already tested software modules with proven performance and reliability. Although it is very common to re-invent the wheel many times, this is certainly not the best possible approach. A pre-requisite for reusing software in aerospace applications is the willingness of airworthiness authorities to accept a reduction of the amount of required testing for those software components, which have already been tested in other projects. This in turn requires the participation of software engineers with sufficient knowledge of available existing software modules early during system design.

In the last few years, a significant change of minds has become visible with respect to re-use of software. Whereas the now outdated 2167A standard [13], which has been used as the basis for many monitoring projects is written in terms of new development, the new 498 standard [14] acknowledges the necessity and advantages of re-using or re-engineering of existing software and provides practical guidelines how to apply the standard to re-used items.

One of the most important verification aims is to proof the correct implementation of the life usage algorithms. Here we followed the route of multi version programming discussed above. One version is the so-called Mission Analysis Program (MAP). This is a program package written in FORTRAN, which can be used for all engine types, for which we have developed monitoring algorithms. It is installed on workstations in the Structural Mechanics department of our company. The main purpose of this program is to analyse missions (synthetically generated profiles as well as flight recorded data) with respect to life consumption and other tasks in connection with engine life management. The MAP also has the functionality to perform the Monte Carlo analysis outlined above.

The other version is a so-called reference program for the specific project. It is written in the project specific programming language (preferably C or Ada). With respect to the life usage algorithms the source code is exactly the same as for the target system, but compiled and run on a workstation or a personal computer (PC).

Both programs are fed with identical input profiles. The complexity of the mission profiles increases in the course of testing. First only simple synthetic profiles (e.g. just an engine acceleration and deceleration) and later on more complex profiles, up to recorded mission data with unusual engine manoeuvring are used.

During integration of the life usage algorithms, extensive tests are performed to check the accuracy of the intermediate results (temperatures, stresses) and of the final results (damage increments) for a representative variety of engine configurations and test data (recorded flight missions). The test method applied is a comparison of the target software results against those of MAP maintained by the Structural Mechanics department. The tests are performed by an independent test team (typically consisting of highly qualified software engineers), whose main task is the automatisation of test procedures. This automatisation is indispensable for limiting test cost, because the initial investment into an optimised test environment pays back several times, when tests have to be repeated due to errors detected or due to necessary design modifications in subsequent development phases. The requirements for accuracy of the on-board implementation of the monitoring algorithms is derived from the attainable accuracy of the monitoring model relative to the FE-calculation of the rotor structures. A characteristic acceptable deviation for temperatures has an order of magnitude of 10 K (for more details refer to [4]). To retain this accuracy also for the implementation within the target system, the deviation between accurately computed results with the monitoring model and its approximation (e.g. using integer arithmetic) in the on-board

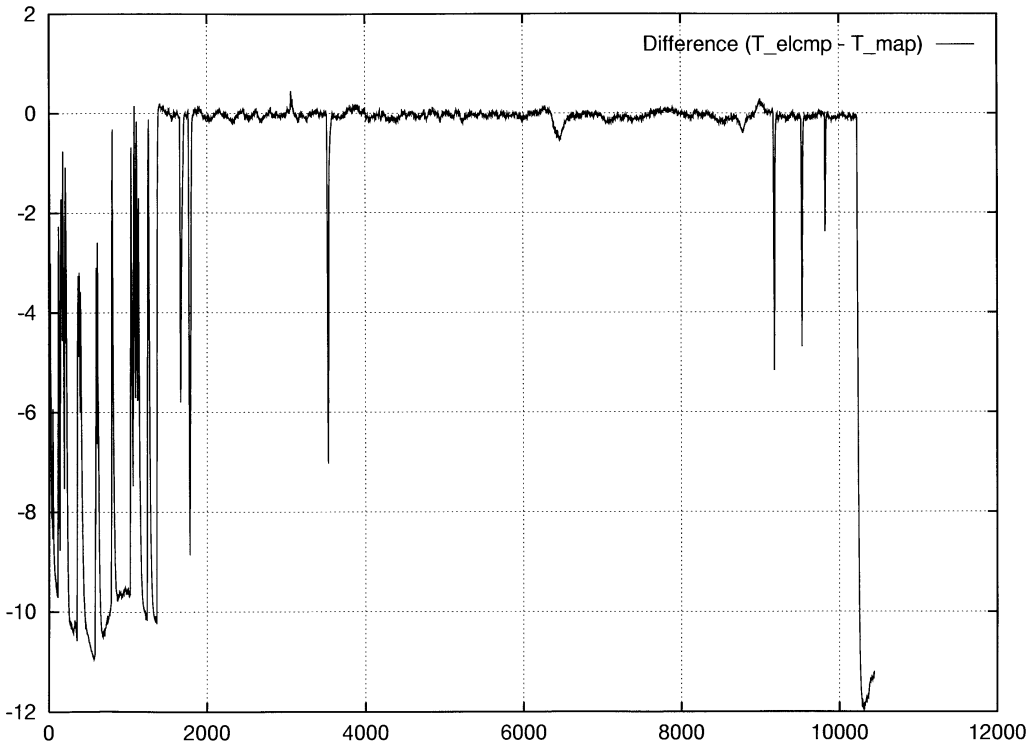


Figure 4: Temperature Difference between MAP and Reference Program Revealing the Programming Error

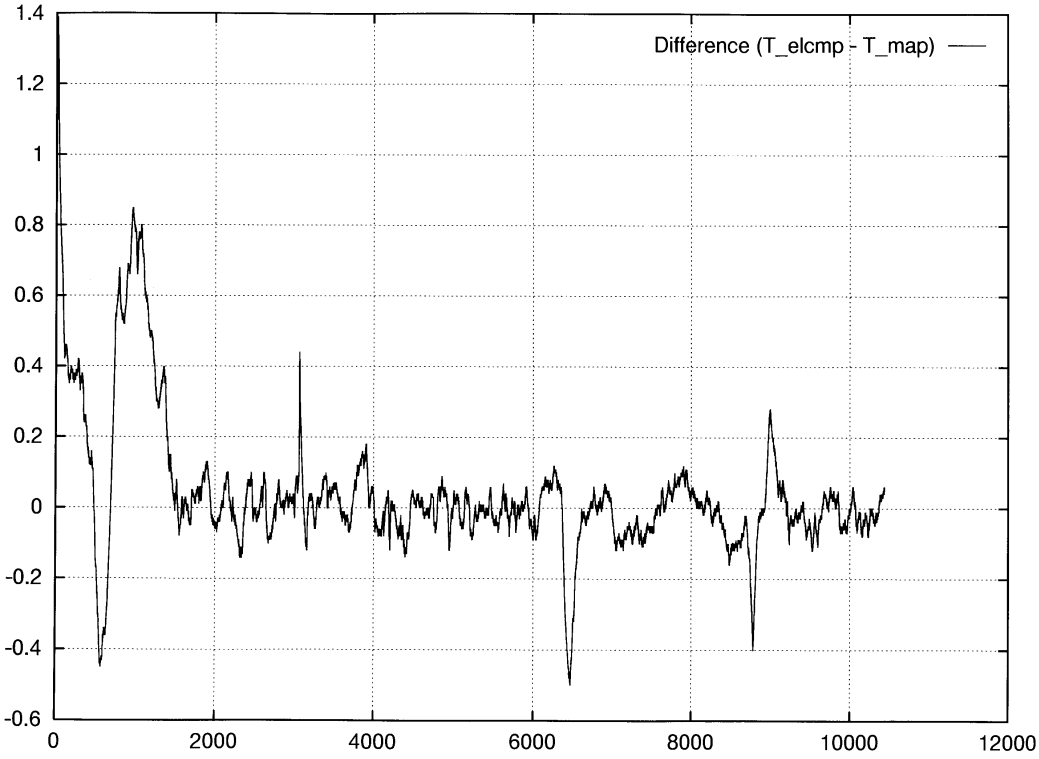


Figure 5: Temperature Difference after Error Fixing (note the different scale)

software is required to be considerably lower (e.g. 1 K).

An impression of the effectiveness of this test procedure can be obtained from the following example. During one of the comparison tests of the current version of the RB199 ELCMP (refer to [8]), a problem report was raised by one of the software testers, indicating a poor correspondence of some stresses on a turbine disk, especially in the idle phases at the beginning and at the end of an engine run. A subsequent analysis by the responsible software designer revealed that the stress calculation itself was correct, but that the deviation was already present in an intermediate result (stationary leading temperatures) not analysed during earlier tests. The difference between the MAP result and the (incorrect) reference software result is shown in Figure 4. It is clearly visible that the difference is very low during most of the flight, but exceeds the acceptable tolerance of ±1 K for long time intervals at the start of the engine run, a few times during the flight and again at the end of the flight. By debugging and code analysis the error was traced back to the computation of stationary leading temperatures, where one special case (interpolation for low spool speeds) contained a programming error. Earlier tests had failed to reveal this error, because due to different behaviour of the engine standards, the corresponding interval had not been hit with the applied test data.

After a correction of the programming error (which in fact resulted from copying a similar piece of code from another module which did not cover the more general situation at the target location) and building a new version of the software, the test was repeated and now the results became very satisfactory and the deviation dropped below the required limit of ±1 K (Figure 5).

Similar comparison tests are made between the reference program and the software integrated in the target system. But here the accuracy requirements are even higher. As the source code is exactly the same, we expect that the results are bitwise identical. Any deviation detected is a reason to investigate for errors.

The necessity to perform comprehensive tests of the internal processing and of the internal interfaces within the on-board components of an engine monitoring system can only be met by providing a hardware test environment as shown in Figure 6.

The general rule is to test each functionality at the highest accessible level and to limit hardware related tests to issues like resource utilisation, real time behaviour or hardware error detection. The final testing of interfaces has also inevitably to be performed with target system components.

There is a significant benefit from performing the major part of software development and testing in a standard computer (mainframe, workstation, PC) environment. The compilation and analysis times are significantly reduced.

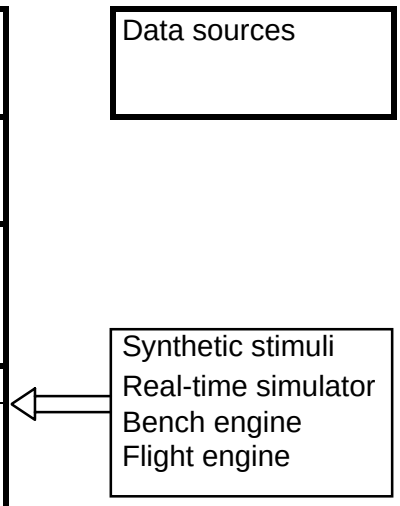
If monitoring systems are to be integrated as subsystems of an engine control system or as part of a general purpose data acquisition system, the availability of the final target hardware may be delayed to a rather late phase of development. In such a case a processor prototype board may be helpful to check the functionality of the target system compiler and the processor load. The additional effort for the adaptation of the prototype hardware development system should be considered.

In addition to a safe and reliable design of the system there should be facilities to support effective testing:

- accessibility of interfaces for software and parameter loading in the aircraft environment
- a way to switch from input signals to recorded digital parameters
- facilities to monitor the system by test protocols or, preferably, ...
- ... cyclic transmission of status information via a serial interface line (not connected on-board)

There may be objections to install software which is not productive for the target application. However, a design (SW or HW) which does not support test and maintenance will retaliate by the time and effort wasted in future trouble-shooting.

Hardware Environment	real-time*) / on-line**)	Development phase *(test goals)*
Host computer / PC Office	no / no	SW development *(algorithms, SW modules, SW sequencing)*
Prototype HW Office/Laboratory	yes / no	Prototype tests / target not yet available *(memory & timing pre-investigations)*
Target System		
Laboratory	yes / no	HW/SW integration / pass-off test *(interfaces, timing)*
Real-time Simulator	yes / yes	System integration *(interfaces, data capture)*
Bench	yes / yes	System integration engine *(engine data capture)*
Aircraft	yes / yes	System integration A/C *(data capture, any other effect: user handling, other system interactions, ...)*



*) real-time: the analysis of an engine run lasts as long as the corresponding engine run

**) on-line: simultaneous analysis during the engine run

Figure 6: Types of Test Environment Used for Life Usage Monitoring System Testing

VALIDATION

Validation of the life usage monitoring system has the aim to demonstrate that the product meets the expectations of the customer. One essential target is to find specification errors (e.g. incorrect presumptions, insufficient description of the physical behaviour of engine and components, unknown aspects of system handling or deviations from assumed aircraft and engine operation). The behaviour of sensors under

operational conditions can also produce unexpected problems (e.g. noise, spikes and drop-outs, sensor drift and total failure). To account for human errors in operating the system, the validation tests should be performed within the productive environment involving the personnel later on responsible the productive utilisation of the monitoring system as also pointed out in [17].

A good means to perform monitoring system validation testing is to carry out flight trials. For these tests it is very important to check the complete data transfer path (sketched in Figure 1). It starts with the engine and aircraft signals (generated by sensors), passes to the on-board processing system and to the internal storage media. The data path includes then on-board displays (such as cockpit, maintenance data panel or connected ground equipment) and a transfer medium to the ground station. The local ground station itself with its periphery as well as data transfer to the remote logistic support system should be scrutinised. But also further data transmissions to spare part management systems, workshops and supporting industry should be checked for data consistency. It should be noticed that most of the interfaces involved are bi-directional and data transfer in both directions must be considered.

When performing flight trials it is very important to have quick access to the simultaneously recorded flight data, because this seems the only way detected problems can be completely investigated (e.g. by interviewing the pilots and operating crew, checking hardware and sensors). Ideally, immediately after the flight the recordings of all input data and the results of on-board data processing are available for being fed into a reference implementation of the monitoring software at a locally available PC. Under such conditions, detected problems can be quickly and easily reproduced and investigated.

If serious problems occur (e.g. with internal interfaces, timing, data transfer), it is advantageous, if the flight recorded data can be re-converted into the original format (as input into the on-board processing equipment) and then fed into a separate prototype box via a special test interface. With an emulator and continuous real time observation of special diagnosis information, it is then possible to reproduce bugs occurred during flight, and to analyse and fix them.

A critical issue for the correct function of any embedded system and also for an on-board engine monitoring system is the timing of processor systems and software during power-on or power-off. If different functions are involved, each function typically performs various self tests, sometimes also including hardware related tests.

A careful design of interfaces, including timing analyses of start-up sequences, is necessary to avoid interference between subsystems during the set-up phase. Interactions between self-tests of connected processor systems tend to produce strange and unpredictable results.

During the flight trials for ELCMP08 in 1991 a situation occurred, where the update of the Data Acquisition Unit (DAU) failed in some rare occasions. The reason was found to be a reset request sent to the engine monitoring processors shortly after power up. This high priority request was accepted even during the self test of RAM (Random Access Memory). A very small portion of RAM is used to store engine configuration data and permanent accounts. As the reset request happened to arrive exactly at the time when the RAM test was checking the cells carrying the engine configuration information, a few storage locations were not properly restored and the subsequent plausibility test of the configuration data reported an update error. If only

RAM working memory used by the lifing algorithms was affected, this memory was initialised properly at engine start, thus hiding the undefined state of a few RAM cells affected by the interrupted self test. Due to their low probability and the different timing behaviour between prototype and production environment, such errors may remain undetected during development and prototype testing.

Even if the malfunction shows up more frequently in the flight trial environment, it may remain hidden for a long time due to the interaction with other system problems or simply by the assumption of the test personnel "We might have done something wrong in the procedure". What would you do, if you have got a brand new electronic equipment with new software, you switch on power, press some buttons following a new procedure you do not fully understand, and a message "Update Error" appears on the display at some point in the procedure? Certainly, your first hypothesis would to assume that you missed something in the procedure. You would decide to switch power off and then on again and to repeat the procedure - and everything runs smoothly. But the explanation was that the timing conditions were different in both attempts and the few RAM cells spoiled by the bug in the memory test had a 5% probability to contain configuration data. When the error message re-appeared 15 or 20 flights later, it was quite natural to assume a handling mistake instead of a real error, which had the consequence for you to fill out a lengthy problem report form.

One general recommendation concluded from this error is to keep the error reporting process as simple as possible and not to overburden the test personnel with bureaucratic procedures, which might give rise to the retention of useful information.

PROCEDURES

Software verification and validation require a number of procedures to be followed. The most important are the software test plan, the software test description and the software test report (for details refer to [16]).

The purpose of the software test plan is to describe the plans for qualification testing of CSCIs and software systems. Encompassed are the description of the test environment including test sites, the software, hardware and firmware items as well as the personnel involved and the schedules for the test activities. The tests to be performed are identified and traced back to the requirements they address.

The software test description serves to describe the test preparations, test cases and test procedures. The preparations include providing specific hardware to be used, switch settings, step-by-step start-up instructions, specific software and associated storage media, software loading instructions, and any other pre-test personnel actions as well as the procedures necessary to perform the test itself. The test cases include pre-requisite conditions, test inputs, expected test results and criteria for evaluating the obtained test results. Test procedures consist of operator inputs and actions, expected results and evaluation criteria for each step, actions to follow in the event of a program stop or indicated errors.

For life usage monitoring, most of the test cases require input profiles which define particular engine operating conditions. The first tests are based the engine design missions (defined as synthetic data profiles). As the algorithms need to cope with all extreme operating conditions defined in the specification, it is necessary to construct a number of specific test cases. Maximum values of spool speeds, temperatures,

pressures as well as maximum rates of change need to be simulated in a lot of combinations which may adversely influence each other. Test cases are also required, which simulate error conditions, in order to check the error response of the monitoring system. Adequate error response is essential, as this may be the only way to provide supporting information for trouble shooting under in-service operating conditions. Additionally, we are faced with the fact that realistic recorded data are not available in early stages of system development. The first realistic data available in a new project stem from engine runs on test bed. The tested engines are normally prototypes with a lot of differences to the final production engine. More realistic data are obtained from engine flight trials which are still performed with prototypes. But also the flight trials do not provide enough information about the later in-service handling. Such realistic data are very important, since they reveal particular engine behaviour peculiarities that might not be covered by the specifications. However, it is tacitly required that also those unexpected conditions are correctly treated by the monitoring software. In this way, the test cases are adapted step by step to the available information. The consequence for testing is that the number of test cases increases and a lot of testing is repeated. In this situation, automation of the test processes helps to reduce test effort and time.

The software test report provides a record of the qualification testing performed with the software system. It contains an overview of the test results including overall assessment, impact of test environment and recommended improvements, further the detailed test results for each test describing problems encountered and deviations from the test cases and procedures. A test log is provided with date, time and location of the test, used hardware and software configurations as well as test activities, personnel and witnesses. Again it safes cost and time if the test documentation is generated directly from the test drivers.

A life usage monitoring system undergoes many modifications, not only during the development phase but also under its entire life cycle. This calls for effective measures to steer the modification process. Means foreseen by the development standards are configuration control, problem reports and design changes.

The activities must cover every problem detected in the components of the system. The items to be traced - including all CSCIs - are typically laid down in the software development plan.

Problem reports give detailed descriptions of the problems detected. Design changes contain a problem analysis and the suggested measures to solve the problem. Both together serve as input to the corrective action system. The corrective action system is a closed-loop system, ensuring that all detected problems are promptly reported and entered into the system, action is initiated on them, resolution is achieved, status is tracked, and records are maintained.

For every change or problem solution, it is also necessary to consider the effort to proof its correctness. In order to reduce the test efforts, very often a number of changes are combined to packages. This is in particular then advisable, if it is to be proven that the overall behaviour of the system is not adversely effected by the modifications in question.

The formal corrective action process should apply only to software products after they are placed under project-level configuration control. Applying this process too early in the development process would lead to an overloading of the corrective action system and could also impede the necessary maturation process, which sometimes involves

frequent specification or design iterations not possible within a rigid framework of a formal process.

Many changes to the monitoring system do not result from insufficient design but from the inability to predict all aspects of the daily work under operational conditions. Thus some changes are initiated by the customer. To get an impression which category of problem may be detected by whom, we have sorted the problem reports and design changes gathered during the most recent update of OLMOS-ELCMP development (see [1, 8]) into a matrix shown in Figure 7.

		Problem detected by								
		Partner Company	Customer	Project Manager	System Engineer	Designer	Tester	Walkthrough	Review	Flight Test
Category involved	Plans			1						
	Specification	1	3		1	2	1		1	
	Algorithm Design						3			
	Interface	1				1	1			
	Hardware	1					1			
	Software Design					3	2			
	Programming					2	5	2	1	
	Portability						2			
	Testability						1			
	Test Design						2			
	Test Environment					1				1
	Evaluation Criteria					1				
	Configuration Control	1				1			2	
	Documentation		1	2	2	3				

Figure 7: Problem Report and Design Change Matrix

CONCLUSION

Verification and validation are activities necessary to establish a proper working aero engine life usage monitoring system. Verification can be characterised as the means to show that the product meets all specification requirements, whereas validation has the purpose to demonstrate that the customer's expectations are satisfied. Most of the tasks within these activities encompass testing. The tests include the accuracy proof of the life usage algorithms, evidence for correct implementation of the functionality into the target system as well as checks for consistent data transfer between all items involved.

In a number of projects we dealt with over the last years, we came to the conclusion that testing of a monitoring system sometimes serves to get to know the operating environment which could not be specified detailed enough in the concept and planning phases.

REFERENCES

[1] J. Broede
Engine Life Consumption Monitoring Program for RB199 Integrated in the On-Board Life Monitoring System
AGARD Conference Proceedings No 448, Quebec, 1988

[2] K. Richter
The On-Board Monitoring System of the MTR 390 Engine
17th International Symposium AIMS, Bonn, 1993

[3] F. Hörl, K. Richter
Monitoring the EJ200 Engine
18th International Symposium AIMS, Stuttgart, 1995

[4] J. Broede, H. Pfoertner
Advanced Algorithm Design and Implementation in On-Board Microprocessor Systems for Engine Life Usage Monitoring
15th International Symposium AIMS, Aachen, 1989

[5] G. Dhondt, W. Möhres
Modelling the Engine Temperature Distribution between Shut Down and Restart for Life Usage Monitoring
16th International Symposium AIMS, München, 1991

[6] J. Broede
Design and Service Experience of Engine Life Usage Monitoring Systems
5th European Propulsion Forum, Pisa, 1995

[7] H. Pfoertner, C. Roß
Preparing Life Usage Monitoring for the Next Decade
18th International Symposium AIMS, Stuttgart, 1995

[8] J. Broede, H. Pfoertner
OLMOS in GAF MRCA Tornado - 10 Years of Experience with On-Board Life Usage Monitoring
33rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit, Seattle, 1997

[9] ISO / IEC 9126
Information technology - Software product evaluation - Quality characteristics and guidelines for their use
International Organisation for Standardisation, International Electrotechnical Commission, 1991

[10] H.-L. Hausen, D. Welzel
Guides to Software Evaluation, Arbeitspapiere der GMD 746
Gesellschaft für Mathematik und Datenverarbeitung mbH, Sankt Augustin, 1993

[11] Software Productivity Consortium
Ada 95 Quality and Style: Guidelines for Professional Programmers
US Department of Defense, Ada Joint Program Office SPC-94093-CMC, 1995

[12] RTCA DO-178B
Software Considerations in Airborne Systems and Equipment Certification
RTCA Inc. SC-167, Washington DC, 1992

[13] Military Standard DoD-STD-2167A
Defense System Software Development, 1988

[14] MIL-STD-498
Software Development and Documentation
US Department of Defense, 1994

[15] MIL-STD-498 Application and Reference Guidebook
US Department of Defense, 1996

[16] Data Item Descriptions (DIDs) for MIL-STD-498
US Department of Defense, 1994

[17] R. Merrits, H. Tanner, M. Lopez-Estrada
Testing Military Software in the Near-Operational Environment
Professional Paper, Naval Air Warfare Center Aircraft Division, Patuxent River, MD, 1995