X-Tools: A Case Study In Building World Class Software

Alan Cooke
ACRA CONTROL

Abstract: X-Tools is a collection of utilities for validation, translation, editing and report generation designed to enable the Flight Test Instrumentation (FTI) community to quickly adopt the XidML 3.0 meta-data standard. This paper discusses the challenges of developing such software that meets the current and future needs of the FTI community, and meets the increasingly high quality standards expected of modern software. The paper first starts by discussing the needs of the FTI community and the specific functional requirements of software. These include the ability to fit in with legacy systems, the ability to handle many tens of thousands of parameters, support for new networked-based technologies and support for hardware from any vendor. The non-functional requirements of FTI orientated software are also described and it is suggested that the key non-functional requirements include testability, modifiability, extensibility and maintainability. Finally, as a case study, the X-Tools from ACRA CONTROL are presented. The paper discusses their design, and the tactics used to meet the functional and non-functional requirements of the FTI industry. The paper then outlines how the rigorous quality standards were met and describes the specific mechanisms used to verify the quality of the software.

Keywords: Software Quality**,** Functional Requirements, Non-functional Requirements, Quality Attributes, Software Product Lines

1. INTRODUCTION

The needs of the FTI community are becoming increasingly sophisticated and demanding. These include the requirement to support new communication and packaging protocols, the ability to handle systems with many tens of thousands of parameters and the need for intelligent system diagnostic and pre-flight tools. Additionally, users sometimes require aircraft to be instrumented using a mixture of legacy data acquisition equipment and the latest cutting edge technologies.

FTI projects are also becoming more varied. FTI vendors need to support a user community that includes projects as diverse as the instrumentation of small fixed wing aircraft, helicopters, un-manned air vehicles and very large aircraft that acquire data using highly sophisticated networking technologies.

As the hardware used in data acquisition becomes ever more sophisticated, so too does the demand on software. The software must not only be able to configure these systems, but also provide an interface that is easy to use and intuitive. Increasingly, the user also has greater expectations of software in terms of quality.

2 WHAT IS QUALITY?

In this paper we categorise the quality of software using the following criteria

- Stability and minimal number of bugs?
- Is the software easy to use?
- Does the software meet the functional requirements?
- Does the software meet the non-functional requirements?

The extent to which software meets these criteria determines the quality of the software.

In addition to these criteria, developers of any software should be able to use objective metrics to measure these quantities.

2.1 STABILITY AND A MINIMAL NUMBER OF BUGS

Modern software users demand that the software they use is stable. It is not acceptable for software to crash regularly and they expect the software to behave in a well-known and predictable way.

Similarly, while it is unreasonable to expect truly bug free software, and at the same time make the software affordable, users have a very low tolerance of bugs.

2.2 EASY TO USE SOFTWARE

There are several criteria that can be used to determine if software is easy to use. It is suggested that the following apply to software orientated to the FTI user community.

- The software should be intuitive. It should closely match the users view of the FTI domain.
- The software should be responsive and not degrade markedly as the number of parameters in the system under definition increases
- Other features that make software more usable include undo, auto-save and change history

2.3 FUNCTIONAL REQUIREMENTS

Functionality is the ability of the software to do what is required of it. The functional requirements determine the set of functions required of the software by the target user community. Specifically, in the case of the FTI user community, the following is expected of software.

- The ability to support tens to many tens of thousands of parameters
- The ability to support both legacy and the latest cutting edge technologies
- Support for multiple FTI vendors

- The software should cover everything in the FTI domain from sensors on aircraft, data acquisition units and bus monitors through to ground station equipment and displays on a computer screen.

Additonally, FTI users generate their configuration data from many sources. Some store this data in large databases, some use dedicated software to generate this data, while others employ a combination of these methods. Furthermore, individual vendors usually use their own proprietary mechanisms for configuring their hardware. With this in mind, it is suggested that the following is also expected of software that is aimed at the FTI user community.

- The ability to store user input in a meta-data format that is capable of describing the system under test. Ideally, the meta-data standard should also be vendor neutral, be an open standard and be capable of growing to meet future needs. One of the few meta-data standards that meet these criteria is XidML [1] [2] [3][4].

2.4 NON-FUNCTIONAL REQUIREMENTS

In developing quality software, it is not enough to just meet the functional requirements, it also necessary to meet other goals such as *reliability*, *scalability*, *dependability* and *extensibility*. These characteristics, sometimes referred to *quality attributes* [8], are crucial to the design of the underlying software architecture. Put simply, if the functional requirements were the sole criteria used in software design then almost any software architecture would suffice.

There are two sets of forces, *internal* and *external*, driving the non-functional requirements. External requirements are primarily driven by the functionality expected of the software but can also include other influences such as, for example, the need for software to meet the standards imposed by a regulatory body. The internal requirements are driven by the concerns of the organisation developing the software.

For a given a set of desired quality attributes that the software should posses, it is necessary to employ a set of techniques, or *tactics* [8], to achieve these goals.

2.4.1 EXTERNALLY DERIVED REQUIREMENTS

In the case of the FTI community, given the functional requirements listed above, a set of quality attributes can be determined.

- **Support for tens to many tens of thousands of parameters**: This requirement demands that the software should be *scalable*.
- **Support for both legacy and the latest cutting edge technologies**: This implies that the software should be easily *extensible* in terms of being able to handle new buses, protocols and so on. For example, by adding new screens customised to a new type of bus. The software should also be *modifiable* in the sense that any changes to one part of the software should not effect the functioning of other parts.

- **Support for multiple FTI vendors**: This again implies that the software should be both *modifiable* and *extensible* with an in-built ability to add new hardware, extra communication protocols and so on without affecting other parts of the system.
- **The software should cover everything in the FTI domain**: This requirement further emphasises the previous requirements of *scalability*, *modifiability* and *extensibility*.

In addition to the quality attributes derived directly from the functional requirements there are also those derived from the non-functional requirements.

- **Stable and minimal number of bugs**: In the software industry, it is generally accepted as best practice to test software as early as possible in the development cycle. In order to achieve this it is necessary to design the software in such a way that it is *testable*.
- **Ease of use**: Among the criteria that can be used to measure *usability* is that the software should be responsive and the responsiveness of the software should not degrade significantly, for example, as the number of parameters in the system increases. This suggests that the software should be highly *scalable*.

2.4.2 INTERNALLY DERIVED REQUIREMENTS

These are the requirements imposed by the organisation developing the software. They often relate to how the software is developed and tested.

Typically, development organisations want software to be developed in the most economically efficient manner, in the quickest time possible and be easy to maintain. These requirements usually equate to the quality attributes of *reusability*, *partitionability* and *maintainability* respectively.

2.5 SUMMARY OF QUALITY ATTRIBUTES OF SOFTWARE FOR FTI COMMUNITY

In summary, the following quality attributes are expected of software written for the FTI community.

- **Scalability**: The ability of the software to handle a greater volume of parameters without significantly affecting the responsiveness of the software.
- **Extensibility**: The ability of the software to be extended in terms of functionality and so on without affecting the behaviour of other parts of the system.
- **Modifiability**: The ability to modify a portion of the software without affecting other parts of the system.
- **Testability:** The ability to test the software being developed, especially early in the development cycle.
- **Usability:** The software is responsive and matches the users view of the FTI domain.

Additionally, there are also the requirements imposed by the developing organisation itself. Typically these include

- **Partitionability**: The ability to partition the development of software so that multiple developers can work on the project at the same time.
- **Maintainability**: The ability to adequately document, modularise and construct the software so as to make it easier to maintain by the developing organisation.
- **Reusability**: The ability to reuse as much code, architecture and other software artefacts as possible in order to save both time and money for the developing organisation.

We will now look at the X-Tools as a case study in how these quality attributes are achieved.

3 CASE STUDY: THE X-TOOLS

3.1 WHAT ARE THE X-TOOLS?

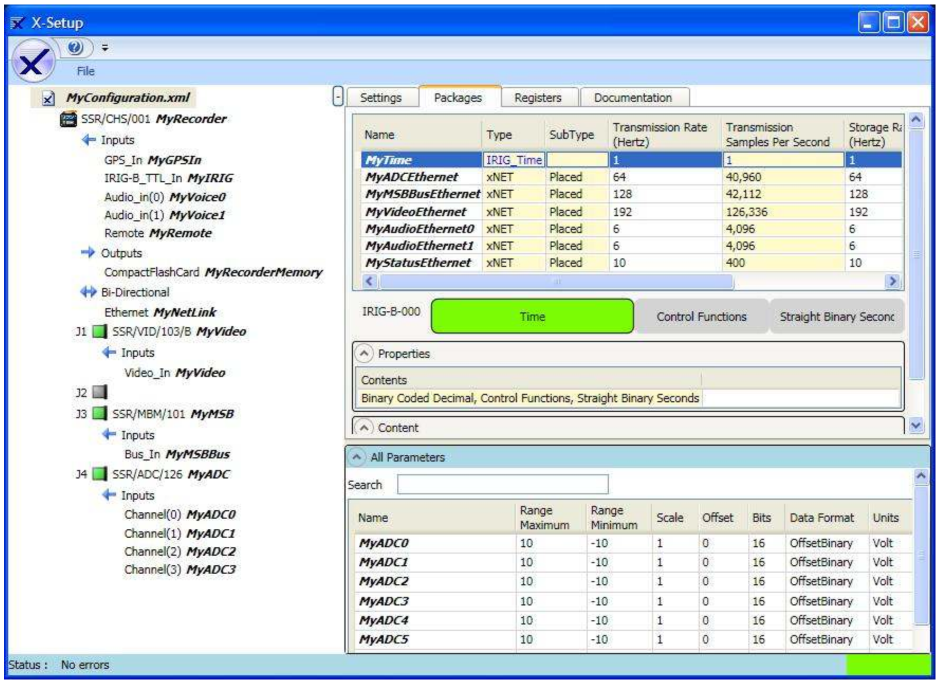
The X-Tools [7] are a collection of freely available tools designed to allow the FTI user community to quickly and easily adopt the XidML 3.0 meta-data standard.

3.1.2 X-SETUP

X-Setup is a XidML native application that can be used to define a complete FTI system. It covers everything in the FTI domain from sensor to screen. Its key features include,

- **Support for up to 100,000 parameters**: X-Setup is designed to be able to handle up to 100,000 parameters.
- **Intuitive System Navigator**: The Navigator provides a natural representation of the FTI system that is being configured. This context sensitive screen allows the user to, for example, view all settings on one screen or to filter what is displayed to a sub set of the data that is of interest to the user.
- **Global Parameter Lists**: The application allows the user to view all parameters, from all sources, that have been defined in the system
- **Component Libraries**: The Palette is a powerful feature that allows the user to create libraries of data that facilitates the reuse of data over and over again. For example, a user may reuse IRIG-106-Chapter-4 PCM frame definitions, custom measurement units or even the content and configuration of an entire ARINC-429 bus.
- **Customised Package Builders:** X-Setup has a powerful package builder that allows the user to create, for example, IRIG-106-Chapter 4 PCM frames, Network packets, MIL-STD-1553 and ARNC-429 messages etc. in an intuitive way. It also includes auto-place functionality that allows to user to place many thousands of parameters in all the supported package types.
- **Integration with XdefML**: X-Setup fully supports XdefML. XdefML is an XML schema that allows FTI vendors to fully specify configuration data for their hardware. X-Setup uses this data to auto-generate screens used to setup hardware and to automatically validate user input.

Figure 1: A screen shot of X-Setup



MyConfiguration.xml
SSR/CHS/001 MyRecorder
Inputs
GPS_In MyGPSIn
IRIG-B_TTL_In MyIRIG
Audio_in(0) MyVoice0
Audio_in(1) MyVoice1
Remote MyRemote
Outputs
CompactFlashCard MyRecorderMemory
Bi-Directional
Ethernet MyNetLink
J1 SSR/VID/103/B MyVideo
Inputs
Video_In MyVideo
J2
J3 SSR/MBM/101 MyMSB
Inputs
Bus_In MyMSBBus
J4 SSR/ADC/126 MyADC
Inputs
Channel(0) MyADC0
Channel(1) MyADC1
Channel(2) MyADC2
Channel(3) MyADC3

Name	Type	SubType	Transmission Rate (Hertz)	Transmission Samples Per Second	Storage Ra (Hertz)
MyTime	IRIG_Time		1	1	1
MyADCEthernet	xNET	Placed	64	40,960	64
MyMSBBusEthernet	xNET	Placed	128	42,112	128
MyVideoEthernet	xNET	Placed	192	126,336	192
MyAudioEthernet0	xNET	Placed	6	4,096	6
MyAudioEthernet1	xNET	Placed	6	4,096	6
MyStatusEthernet	xNET	Placed	10	400	10

IRIG-B-000 | Time | Control Functions | Straight Binary Seconc

Properties

Contents
Binary Coded Decimal, Control Functions, Straight Binary Seconds

Content

All Parameters

Search

Name	Range Maximum	Range Minimum	Scale	Offset	Bits	Data Format	Units
MyADC0	10	-10	1	0	16	OffsetBinary	Volt
MyADC1	10	-10	1	0	16	OffsetBinary	Volt
MyADC2	10	-10	1	0	16	OffsetBinary	Volt
MyADC3	10	-10	1	0	16	OffsetBinary	Volt
MyADC4	10	-10	1	0	16	OffsetBinary	Volt
MyADC5	10	-10	1	0	16	OffsetBinary	Volt

Status : No errors

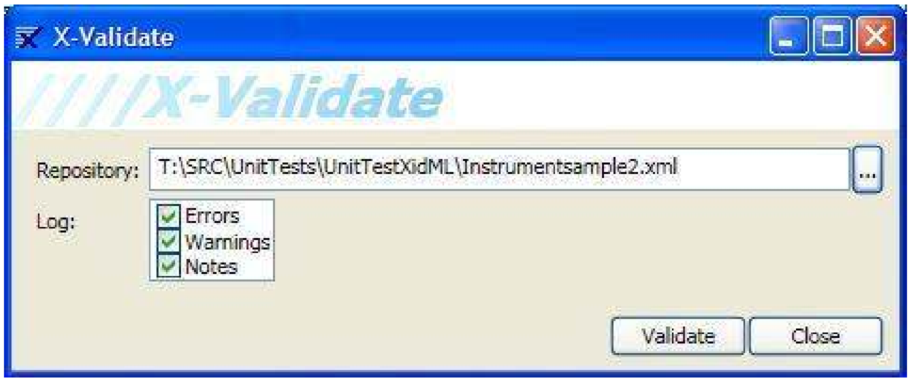
3.1.3 X-VALIDATE

This application is used to verify and validate the data contained in a XidML-3.0 file. It does this by

- Checking that the file is well formed
- Checks that the file is valid with respect to the XidML-3.0 schema
- Uses any XdefML files for instruments defined in the XidML file to validate the configuration data for that instrument
- Produces a report that indicates any failures

The application can be run in full display mode via a user interface or in batch mode from the command line.

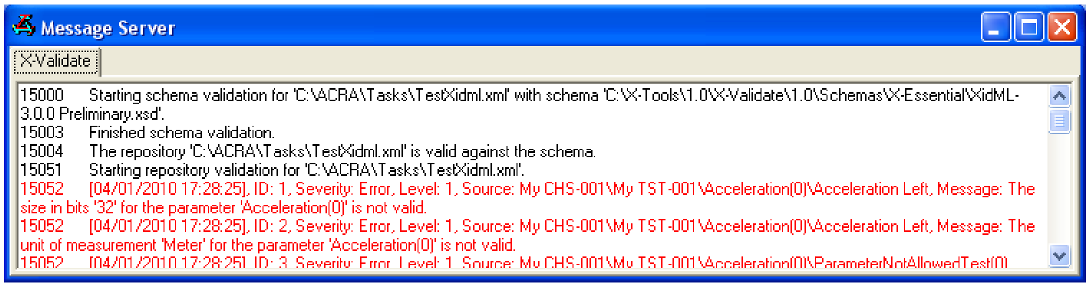
Figure 2: A screen shot of X-Validate



Repository: T:\SRC\UnitTests\UnitTestXidML\Instrumentsample2.xml

Log: ✓ Errors ✓ Warnings ✓ Notes

Figure 3: Example output from X-Validate

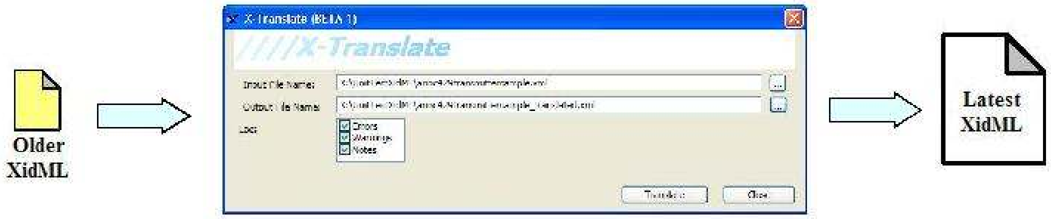


```
15000     Starting schema validation for 'C:\ACRA\Tasks\TestXidml.xml' with schema 'C:\X-Tools\1.0\X-Validate\1.0\Schemas\X-Essential\XidML-
3.0.0 Preliminary.xsd'.
15003     Finished schema validation.
15004     The repository 'C:\ACRA\Tasks\TestXidml.xml' is valid against the schema.
15051     Starting repository validation for 'C:\ACRA\Tasks\TestXidml.xml'.
15052     [04/01/2010 17:28:25], ID: 1, Severity: Error, Level: 1, Source: My CHS-001\My TST-001\Acceleration(0)\Acceleration Left, Message: The
size in bits '32' for the parameter 'Acceleration(0)' is not valid.
15052     [04/01/2010 17:28:25], ID: 2, Severity: Error, Level: 1, Source: My CHS-001\My TST-001\Acceleration(0)\Acceleration Left, Message: The
unit of measurement 'Meter' for the parameter 'Acceleration(0)' is not valid.
15052     [04/01/2010 17:28:25], ID: 3, Severity: Error, Level: 1, Source: My CHS-001\My TST-001\Acceleration(0)\ParameterNotAllowedTest(0)
```

3.1.4 X-TRANSLATE

This application can be used to translate XidML-2.41 files to XidML-3.0 files. This application can also be run in full display mode or in batch mode from a command line.

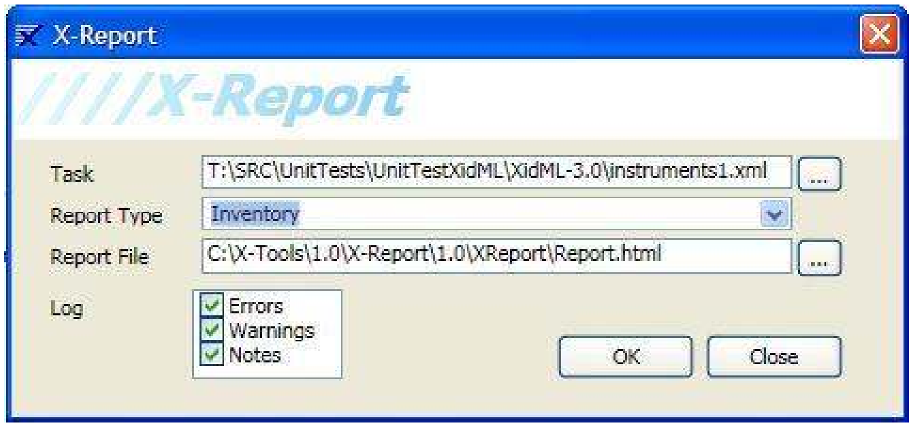
Figure 4: A screen shot of how X-Translate works



3.1.5 X-REPORT

This application takes a XidML file as input and generates one of a number of HTML reports

Figure 5: A screenshot of X-Report



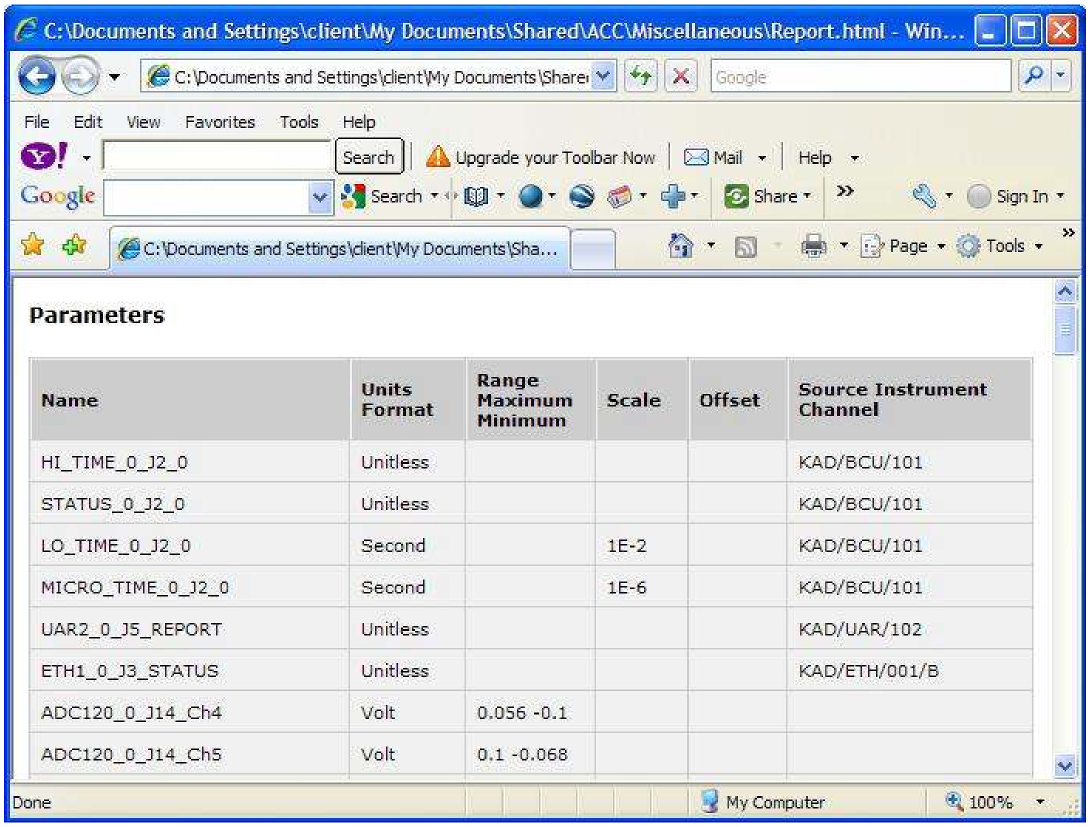
Task: T:\SRC\UnitTests\UnitTestXidML\XidML-3.0\instruments1.xml

Report Type: Inventory

Report File: C:\X-Tools\1.0\X-Report\1.0\XReport\Report.html

Log: ✓ Errors ✓ Warnings ✓ Notes

Figure 6: An example of a report generated by X-Report



C:\Documents and Settings\client\My Documents\Shared\ACC\Miscellaneous\Report.html - Win...
File Edit View Favorites Tools Help
Search
Upgrade your Toolbar Now
Mail
Help
Share
Sign In
Page
Tools
Done
My Computer
100%

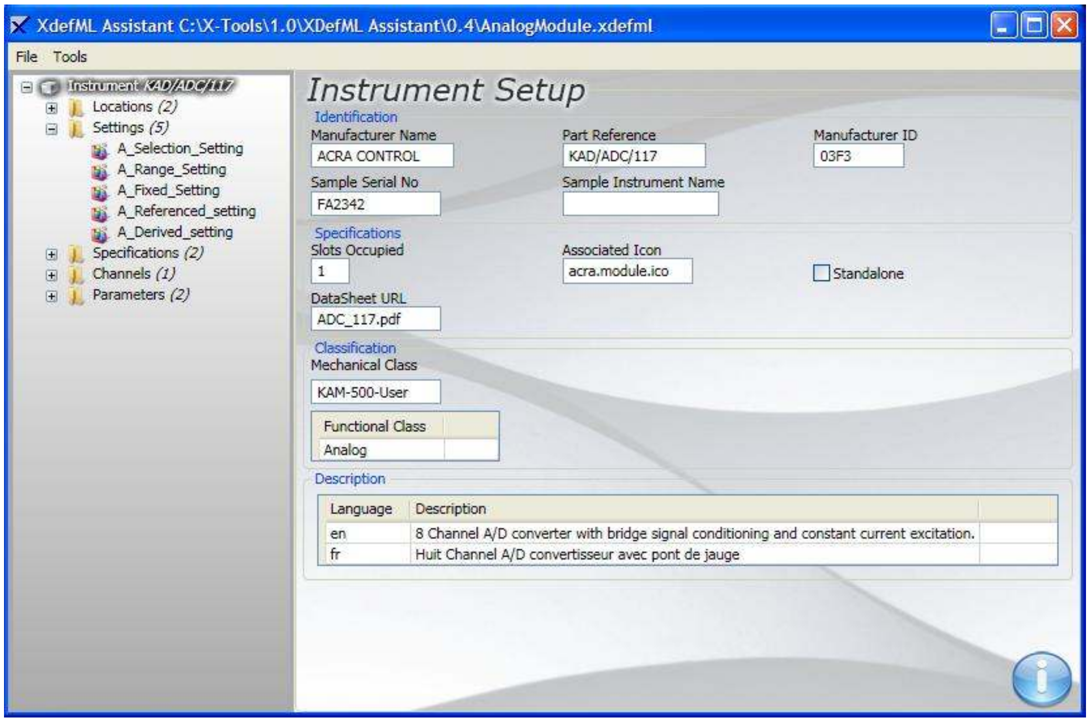
Parameters

Name	Units Format	Range Maximum Minimum	Scale	Offset	Source Instrument Channel
HI_TIME_0_J2_0	Unitless				KAD/BCU/101
STATUS_0_J2_0	Unitless				KAD/BCU/101
LO_TIME_0_J2_0	Second		1E-2		KAD/BCU/101
MICRO_TIME_0_J2_0	Second		1E-6		KAD/BCU/101
UAR2_0_J5_REPORT	Unitless				KAD/UAR/102
ETH1_0_J3_STATUS	Unitless				KAD/ETH/001/B
ADC120_0_J14_Ch4	Volt	0.056 -0.1			
ADC120_0_J14_Ch5	Volt	0.1 -0.068			

3.1.5 XDEFML ASSISTANT

The XdefML Assistant is an application for creating and editing XdefML files. It also features a common settings dictionary, built in defaults and validation.

Figure 7: A screen shot of the XdefML Assistant



Manufacturer Name	ACRA CONTROL
Part Reference	KAD/ADC/117
Manufacturer ID	03F3
Sample Serial No	FA2342
Sample Instrument Name	
Slots Occupied	1
Associated Icon	acra.module.ico
DataSheet URL	ADC_117.pdf
Mechanical Class	KAM-500-User
Functional Class	Analog

Language	Description
en	8 Channel A/D converter with bridge signal conditioning and constant current excitation.
fr	Huit Channel A/D convertisseur avec pont de jauge

3.2 HOW THE REQUIREMENTS WERE ACHIEVED

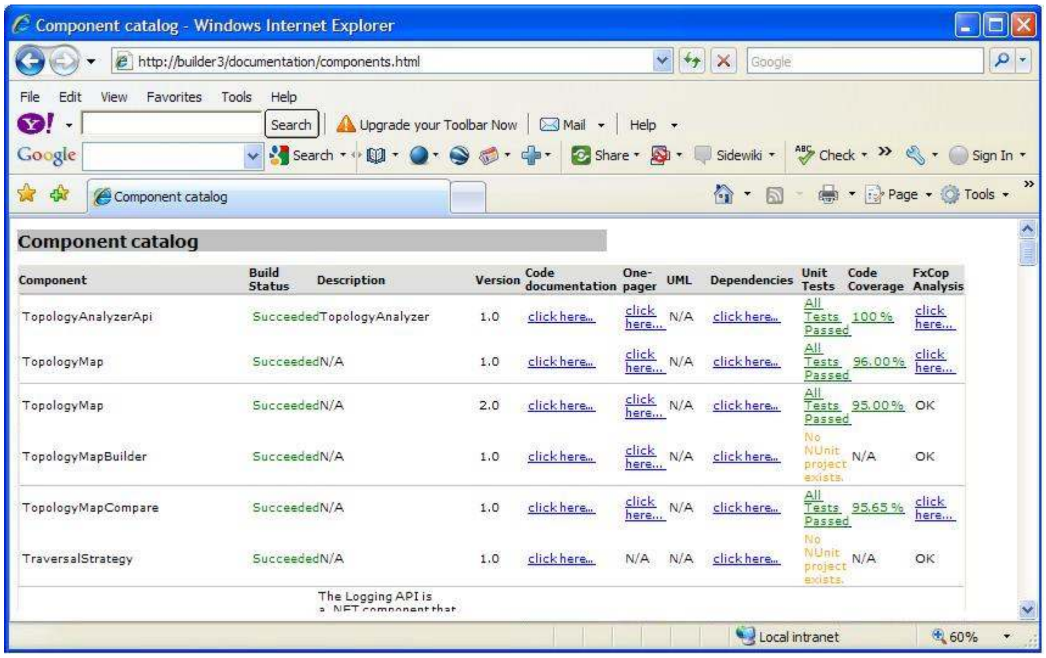
The following sections list how the requirements, both functional and non-functional, were achieved.

3.2.1 STABILITY AND A MINIMAL NUMBER OF BUGS

These criteria were met primarily using the following techniques and methods.

- Each component, or module, was unit tested to greater than 95% code coverage. These tests were fully automated and integrated directly into the software build system. Results were also verified automatically as part of the build system and published on the internal intranet (See Figure 9 below). All of this could be achieved because the software was designed to be *testable* from the start.
- Component integration tests were also written and run automatically as part of the build process. All regression tests were also verified automatically as part of the build system (See Figure 8 below). All failures are automatically reported by email. These tests helped to ensure that all components interact with each other in a predictable way. Again, all this is facilitated by the requirement that the software be designed to be both *testable* and *maintainable*.
- The software was also released in a phased manner as a series prototype, alpha and beta releases. This helped to ensure that bugs were found before the final release of software.

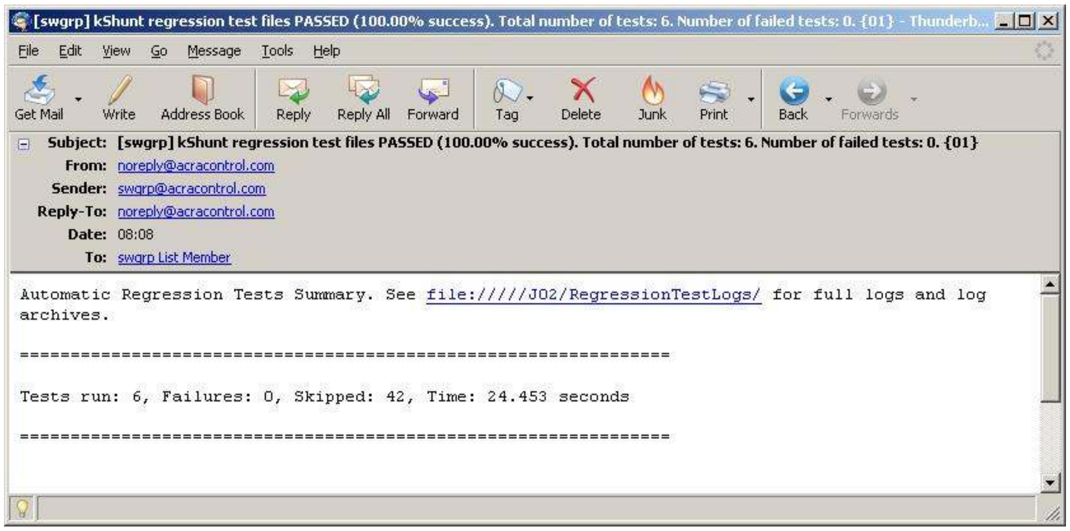
Figure 8: A screenshot of the Component Catalogue from ACRA Controls intranet



Component catalog

Component	Build Status	Description	Version	Code documentation	One-pager	UML	Dependencies	Unit Tests	Code Coverage	FxCop Analysis
TopologyAnalyzerApi	Succeeded	TopologyAnalyzer	1.0	click here...	click here...	N/A	click here...	All Tests Passed	100 %	click here...
TopologyMap	Succeeded	N/A	1.0	click here...	click here...	N/A	click here...	All Tests Passed	96.00 %	click here...
TopologyMap	Succeeded	N/A	2.0	click here...	click here...	N/A	click here...	All Tests Passed	95.00 %	OK
TopologyMapBuilder	Succeeded	N/A	1.0	click here...	click here...	N/A	click here...	No NUnit project exists.	N/A	OK
TopologyMapCompare	Succeeded	N/A	1.0	click here...	click here...	N/A	click here...	All Tests Passed	95.65 %	click here...
TraversalStrategy	Succeeded	N/A	1.0	click here...	N/A	N/A	click here...	No NUnit project exists.	N/A	OK
		The Logging API is								

Figure 9: A screenshot of auto-generated results of a regression test



Subject: [swgrp] kShunt regression test files PASSED (100.00% success). Total number of tests: 6. Number of failed tests: 0. {01}
From: noreply@acracontrol.com
Sender: swgrp@acracontrol.com
Reply-To: noreply@acracontrol.com
Date: 08:08
To: swgrp List Member

```
Automatic Regression Tests Summary. See file://///J02/RegressionTestLogs/ for full logs and log
archives.

================================================================

Tests run: 6, Failures: 0, Skipped: 42, Time: 24.453 seconds

================================================================
```

3.3.2 EASE OF USE

As previously mentioned, the software was released in a phased manner as part of a series of prototype and beta releases. This ensured that user feedback was obtained early and that, where necessary, adjustments were made to both the look and feel of the software and the underlying functionality prior to the final release.

3.2.3 FUNCTIONAL REQUIREMENTS

X-Setup, and the X-Tools in general, meet all the functional requirements of software designed for the FTI community.

- The ability to support tens to many tens of thousands of parameters. X-Setup is designed out-of-the-box to handle many thousands of parameters. One of the reasons it can do this is because it is designed to take advantage of multiple processors.
- X-Setup is XidML native which makes it possible to support both legacy and new emerging technologies.
- Through the use of XidML, and the associated XdefML schema, it is possible to support the configuration of hardware from multiple FTI vendors
- X-Setup is designed to cover everything in the FTI domain from sensor to screen. This is made easier because X-Setup is XidML native.

3.3.4 NON-FUNCTIONAL REQUIREMENTS

Given the set of quality attributes discussed earlier the following briefly outlines how they were achieved in the X-Tools collection of tools.

Scalability

- X-Setup employs a multithreaded architecture and is designed to take advantage of multiprocessor and multi-core computer architectures
- The context sensitive user interface in X-Setup has been designed to allow the user to view a large number of parameters, hardware settings and so on, or to focus on specific sub-sets of this data.
- X-Setup is designed to allow users to construct large package definitions such as IRIG-106 Chapter 4 PCM frames and network-based transport protocols and so on,
- One of the defining features of X-Setup is its ability to work with user-defined libraries. This allows users to define large portions of their system in libraries and to re-use them over and over again, for example all messages and other characteristics of an ARINC-429 bus.

Extensibility

X-Setup is designed to be extensible in a specific number of ways. For example,

- It is possible to add new screens to the user interface to facilitate the introduction of new types of hardware or hardware from different vendors without having to alter the underlying architecture.
- It is possible to add new screens to the user interface to facilitate the introduction of user interfaces for new transmission or storage packages without having to alter the underlying architecture.
- The architecture allows for the introduction of plug-in tools without altering the underlying architecture.
- X-Setup is XidML native and XidML itself is designed to be extensible.

Modifiability

X-Setup is designed so that modifications can be made to one part of the system without affecting either the underlying architecture or other components at make up the software. For example,

- All GUI elements implement a common set of interfaces and employ the same communication mechanism, making it possible to replace or modify any of the user interfaces that make up the user interface without any other part of the software being affected
- The underlying architecture is modularised with each component being loosely coupled to the rest of the application via a common communication mechanism making it possible to change the implementation details of key components without any other part of the software being affected.
- All modules are unit tested to greater than 95% unit test coverage. Any change to the expected behaviour caused by modification to individual components is flagged almost immediately

Testability

The X-Setup software is designed from the start to be testable. This is achieved by

- Modularising the code into semantically coherent components (See Figure 8 above).
- Coding to interfaces. This allows tests based on behaviour to be written and automated using standard unit testing software (See Figure 8 above).
- Because the software is modular in nature it allows discrete subsets of the final design to be written and tested early in the development process.

Usability

X-Setup is designed with usability to the fore. In particular,

- The underlying architecture is multithreaded and to take advantage or multi-core and multiprocessor computer architectures. This leads to a greater responsiveness in the software, especially in the user interface
- The underlying architecture allows for the addition of wizards and smart package builders.

Partitionability

X-Setup is completely modularised (See Figure 8 above) and thus allowed for multiple developers to work on its development in parallel.

Maintainability

X-Setup and the X-Tools in general are designed to be easy to maintain. Specifically,

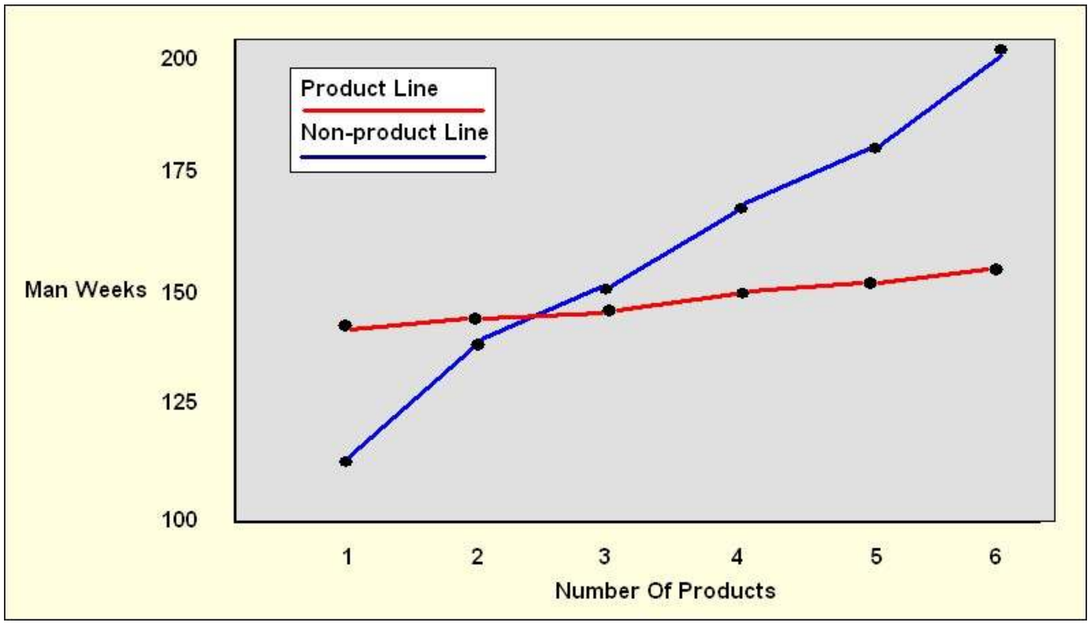
- All of the X-Tools are modular (See Figure 8 above) in nature. This allows problems in individual parts of the software to be localised and fixed in isolation.
- The software is designed to be testable, which facilitated the introduction of automated unit tests. This means that it is less likely for bugs to be introduced inadvertently during a bug test or maintenance phase.
- Because the software is completely modular it is easier to document and thus reduces the learning curve for those tasked with fixing bugs or modifying individual components.
- Components are versioned individually which means that when major changes are made they are easier to target to specific products.

Reusability

From the very start, X-Setup in particular, was designed with re-use in mind. Specifically, X-Setup is just one of a suite of products developed by ACRA Control. This product suite includes DAS Studio, Recorder Studio, Network Studio, GS Studio and BitSynch Studio.

- All of these products use practically the same set of components.
- A *product line* [9] approach was used in the development of these tools. This approach ensured that not only individual components would be re-used but also the architecture and other software artefacts such as test plans and user documentation
- This approach effectively meant that the total cost of developing the software was up to three times cheaper [10], in terms of man-hours, than developing each product individually.

Figure 10: A graph of relative costs of product line versus a non-product line approach



3.3.5 SUMMARY OF TECHNIQUES AND METRICS

The following table summarise the techniques used to achieve the desired quality attributes. It also lists some of the metrics used to verify that the quality attributes were achieved.

Table 1: A summary of the tactics used to meet the desired quality attributes for the X-Tools along with some sample metrics used to verify the design.

Quality Attribute	Examples of Tactics Used	Example Metrics
Scalability	• Take advantage of multiple processors	• Can load 200,000 in less than 30 seconds • Table control can load 1,000,000 parameters in approximately 15 seconds
Extensibility	• Use of XdefML files • XidML native • Predefined plug-in points • Loosely coupled components communicating via a standard communication protocol. • Standardised interfaces	• Can add full user interface and validation support for new analogue to digital modules in one hour • Can add a plug-in tool in seconds
Modifiability	• Modularised architecture • Code to interfaces • Use of XdefML files	• Can change a range for an instrument setting in seconds without the need for recompilation
Testability	• Modularised architecture • Code to interfaces • Use of C# facilitated adoption of automated unit tests	• Each component has its own set of automated unit tests • Test coverage greater than 95% • Published automatically as part of the build process
Usability	• Take advantage of multiple processors • Full drag and drop support • Built in support for undo, redo, change history, auto-save and wizards	• Usability verified as part of beta program • Can auto-place many thousands of parameters to an IRIG-106-Chapter 4 PCM frame in seconds
Partitionability	• Architecture composed of semantically coherent modules • Components loosely coupled from each other	• At height of development, up to ten people worked on project at the same time
Maintainability	• Components individually versioned • Components individually documented • Sample application per component	• Code level documentation auto-generated and published after each build • All UML automatically published on internal intranet
Reusablity	• Modularised architecture • Product line approach	• Greater than 90% of components used in six or more products • Basic test plans and frameworks used in six different products

CONCLUSIONS

This paper discussed the concept of quality with respect to software. The paper suggested that, in addition to minimising the number of bugs and making sure that the software is stable, that both the functional and non-functional requirements must be also be met. The paper then discussed what the functional and non-functional requirements are for software written for the FTI community. It was suggested that the key non-functional requirements, or quality attributes, of any software for the FTI community are *scalability*, *extensibility*, *modifiability*, *usability*. Other typical quality attributes include partitionability, testability, maintainability and reusability. The paper then presented the X-Tools and outlined their functionality. Finally, the paper then described how both the functional and non-functional requirements were met and verified in the development of the X-Tools.

REFERENCES

[1] Alan Cooke, Diarmuid Corry: "XidML: A Global Standard for the Flight Test Community", ETTC Proceedings, 2004
[2] Diarmuid Corry: "XidML 2 Years On"
[3] Diarmuid Corry: "Unleashing the Power of XidML", ITC Proceedings, 2007
[4] Alan Cooke: "Using XML in an FTI Environment", ETC Proceedings, 2008
[5] World Wide Web Consortium, http://www.w3.org/
[6] XidML website, http://www.xidml.org/
[7] X-Tools on the XidML website http://xidml.org/home/forum/tools
[8] Bass, Clements, Kazman. *Software Architecture In Practice*. ISBN 0-321-15495-9.
[9] Clements, Northrup. *Software Product Lines*. ISBN 0-201-70332-7
[10] "Structured Intuitive Model for Product Line Economics": https://simple.sei.cmu.edu/