

2004P0022

# **Accelerated Insertion of Materials – Composites (AIM-C)**

## **Software Component Delivery Requirements**

**Report No. 2004P0022**

V\_2.0.0  
May 07, 2004

Boeing Originator – Original Issue  
Gail L. Hahn  
gail.l.hahn@boeing.com  
Boeing Phantom Works  
St. Louis, MO 63166

Primary Author: Dr. George Orient  
Boeing Rocketdyne  
Canoga Park, CA

Copyright 2004 by The Boeing Company. Published by DARPA with permission.

The original issue of this document was jointly accomplished by Boeing and the U.S. Government under the guidance of NAVAIR under N00421-01-3-0098, Accelerated Insertion of Materials – Composites.

Approved for Public Release, Distribution Unlimited

## Table of Contents

Foreword .....	3
1. Scope of Requirements .....	4
2. Version Tag .....	6
3. Centralizing Environmental Settings .....	7
4. Module Delivery Process .....	9
4.1. Delivery Director Structure .....	10
4.1.1. Directory <b>build</b> .....	10
4.1.2. Directory <b>data</b> .....	11
4.1.3. Directory <b>docs</b> .....	11
4.1.3.1 Directory <b>docs/api</b> .....	12
4.1.3.2 Directory <b>docs/CCB</b> .....	12
4.1.3.3 Directory <b>docs/implementation</b> .....	12
4.1.3.4 Directory <b>docs/theory</b> .....	14
4.1.3.5 Directory <b>docs/user</b> .....	16
4.1.4. Directory <b>env</b> .....	16
4.1.5. Directory <b>Excel</b> .....	16
4.1.6. Directory <b>exe</b> .....	18
4.1.7. Directory <b>lib</b> .....	19
4.1.8. Directory <b>QA</b> .....	19
4.1.8.1. Directory <b>QA\OS\platform\case_name\input</b> .....	19
4.1.8.2. Directory <b>QA\OS\platform\case_name\output</b> .....	19
4.1.8.3. Directory <b>QA\os\platform\case_name\work</b> .....	19
4.1.9. Directory <b>src</b> .....	19
4.1.10. Directory <b>Validation</b> .....	20
5. Template Delivery Process .....	21
5.1 Creating RDCS Batch File .....	21
5.2 Configuration Control of Tools in Functional Models .....	23
5.3 Quality Assurance – Verification of the Math Model Outside of RDCS .....	23
5.4 Directory Structure .....	24
5.4.1 Directory <b>docs</b> .....	25
5.4.2 Directory <b>QA-Math_Model</b> .....	25
5.4.3 Directory <b>QA-Design_Process</b> .....	25
5.4.4 Directory <b>rdcs</b> .....	25
6 The Configuration Control Process .....	25
6.1 CVS – Concurrent Versioning System Overview .....	25
6.2 Tagging the Repository .....	26
7 The Change Control Process .....	27
7.1 The Change Request .....	28
7.2 The Software Release Process (SRP) .....	28
References .....	30

## Foreword

The benefits of integrated product definition teams to balance requirements from multiple sources have been known for years. Each functional expert has knowledge, analysis, and test techniques that are not readily accessible or understood by others. Trade studies and design optimizations are traditionally run function by function with a summation of benefits and shortcomings when all the players come to the table for the final review. What would happen if the functions could be optimized simultaneously? What would happen if the materials focal could see the impact of a change on a structural analysis? What would happen if the structural analyst could trade design considerations with manufacturing defect probabilities? What would happen if the uncertainties associated with analytical techniques and test methods were dissected, understood, and tracked?

DARPA devised the Accelerated Insertion of Materials initiative to answer these questions by challenging Materials, Manufacturing, Structures, Math and Computing personnel to integrate the best of the tools and methodologies of their trades. A Boeing led team and the U.S. Government jointly accomplished “Accelerated Insertion of Materials – Composites” Phase 1 under the guidance of NAVAIR as part of the DARPA sponsored Accelerated Insertion of Materials (AIM) initiative. Dr. Leo Christodoulou, the DARPA Program Manager, and Dr. Ray Meilunas, the NAVAIR technical agent, led the effort. The AIM-C technical team was led by Gail Hahn, Dr. Karl M. Nelson, and Charles Saff of Boeing.

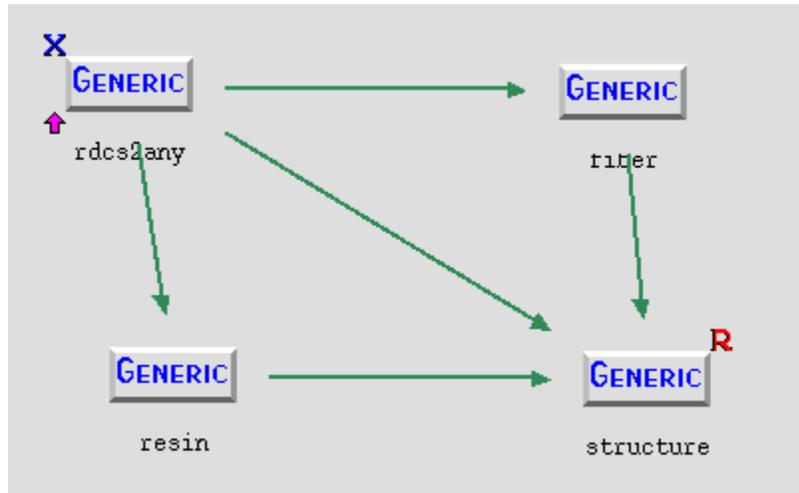
The Accelerated Insertion of Materials – Composites (AIM-C) program established a methodology to evaluate historical roadblocks to effective implementation of composites materials and developed a process to eliminate these roadblocks via knowledge, analysis, and test to mature the material/process/design knowledge base for successful qualification and certification. The approach and analytical tools foster integrated technology/product development teams. Uncertainty is addressed through identification and management of error and application of statistical and probabilistic approaches to facilitate an application solution that is robust (insensitive to known variations).

This document provides software component delivery requirements for anyone interested in conformance to the AIM-C software system as well as the documentation required for a software component to be considered for integration into the system.

## 1. Scope of Requirements

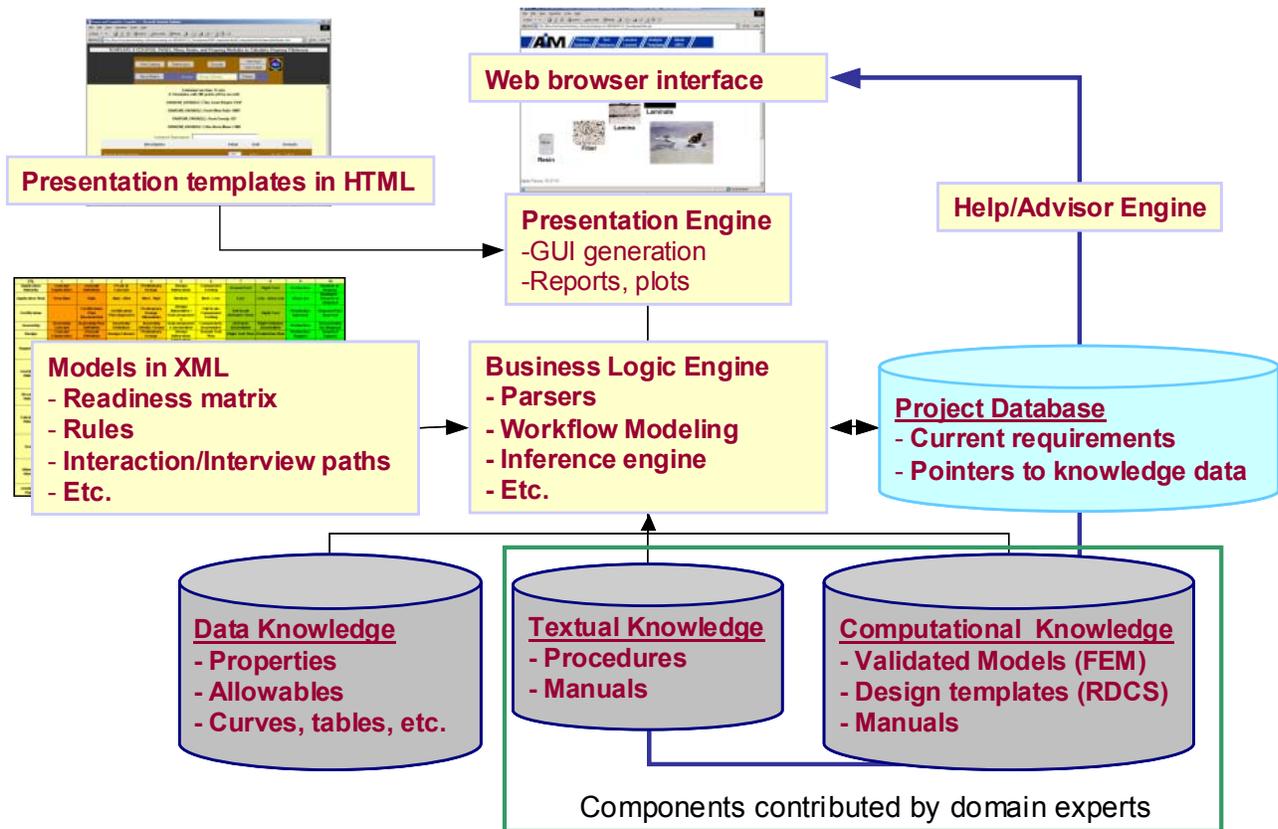
The AIM-C methodology is implemented in the AIM-C system at two levels of integration. The system software is created by the integration team, whereas physics based models, computational design tools and textual knowledge elements are generated by domain experts.

In this context AIM-C modules are defined as an assemblage of mathematical expressions describing relationships between input and output values compiled into a single code. The input of a module is a set of files that are invariant with respect to the design instance (parametric CAD models, compiled mathematical solutions methods, etc.), and a set of variables defining the state of an instance of the module. A template consists of a network of Modules called the Math Model illustrated in Figure 1, and instances of design processes whose goal is to exercise the Math Model to extract specific high level information, such as characterization of the variability for a set of system responses or definition of an optimum solution that satisfies design constraints. The Math Model defines how the individual Modules are interconnected and what type of information flows from one Module to another.



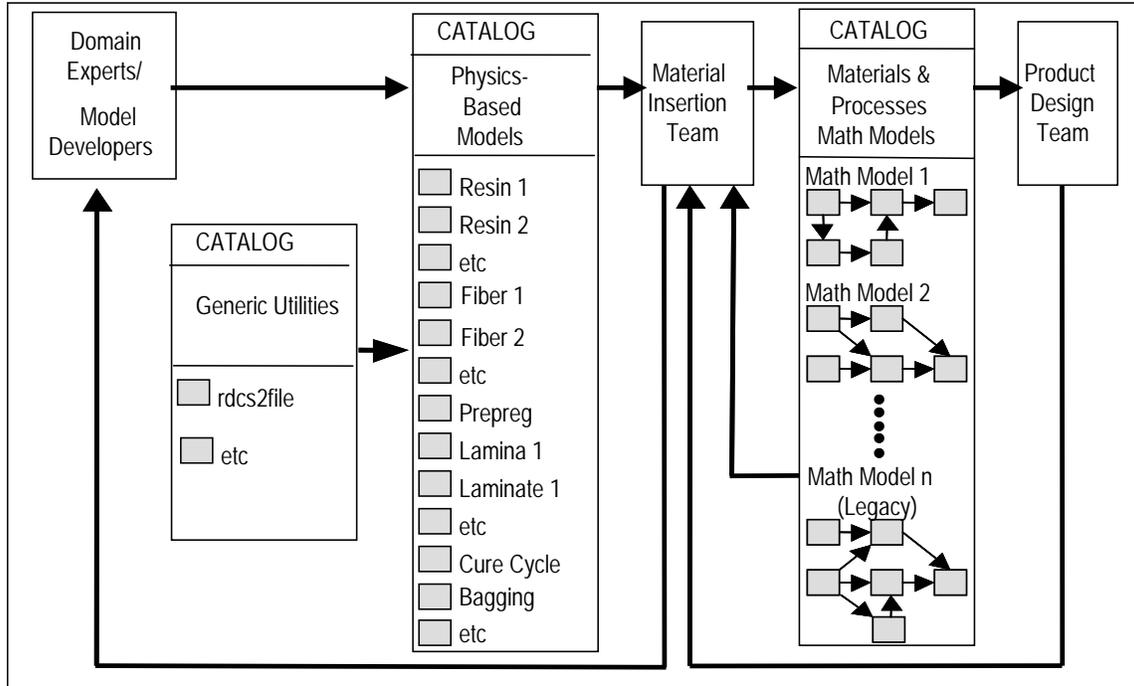
**Figure 1 Math Model example**

The concepts of Modules, Math Model, and design processes are independent of the integration platform. AIM-C employs RDCS, the Robust Design Computational System to implement the specific design templates. RDCS has facilities to characterize variables, their bounds and distributions, and to define and document the Math Model and to define and document the Design Processes. The RDCS integration platform is documented in Reference [1].



**Figure 2 Scope of the current requirements document**

Figure 2 illustrates the scope of the current document. While all levels of the AIM-C system encapsulate material insertion knowledge, certain types of knowledge, such as maturity in the form of readiness levels require documenting the result of conformance assessment and committal in a form recognized by the system. Other methodologies and knowledge, such as modules, templates, and documents are created by the experts and submitted to the Configuration Control Board for approval and integration into the system. As illustrated in Figure 3, the AIM-C computational software catalog housing these contributions is partitioned into three areas: Utilities, Modules and Templates. The difference between a Utility and Module is that utilities typically perform generic tasks not related to a physical discipline.



**Figure 3 AIM-C Computational tools catalog**

The current document describes requirements for the computational tools components.

**Requirement - Modules, templates and documents need to be delivered in a manner that requires no modification other than exposing template control parameters to the system by the integration team.**

This requirement assures that no inadvertent change is made to these knowledge elements, and it also paves the way toward automated knowledge submission in the future.

Throughout its development, each contributed component goes through configuration control administered by the Configuration Control Board (CCB).

## 2 Version Tag

The purpose of the version tag is to uniquely identify a configuration, which may be a physics-based module, a template, or a document. In conjunction with proper practices of version control described in more detail in Sections 6 and 7, this assures that a known configuration can always be retrieved from the repository. For the sake of requirement definition, the exact steps in configuration control are less important than the fact that each work article must have an associated version tag. The version tag has the following format:

*V\_Major.Minor.Fix*

For software the following criteria are applied to define elements of the version tag:

*Major:* Fundamentally new physics, or improved methodology

*Minor:* No fundamental change in capability. Input and/or documentation may change. Numerical results may be affected

*Fix:* Layout change, improved user interface, but no change in the number or the type of inputs, or their numerical values or other information content.

Recognizing that configuration control is not done manually through file naming conventions and reliance on date stamps, it is not required that file names are tagged with the version. In fact, it is preferable to carry a generic file name so that automated tools do not have to be modified when a new release of a component such as data or a script become available.

***Requirement - Version tag must be included in documents in the title and on every page in the rightmost edge of the page footer section.***

***Requirement - If files are tagged with the version tag, a properly formed version tag must be used (V\_Major.Minor.Fix).***

***Requirement – There may be no references to date in the file names. The version tag in the repository has a date associated with it, and re-stating the date is extraneous and a source of error.***

*Recommendation – To simplify tool scripts, avoid adding the version tag to file names for files that are used by tools (input files, tool configuration files, source code, etc). Only tag Word, PowerPoint and PDF file names. Remember, the ultimate authority on version lies with the versioning system.*

### 3 Centralizing Environment Settings

In order to assure easy porting of the AIM-C system between different environments, all configuration settings must be centralized. In other words, no references to users' home directories, paths to tools or other environment settings should be used in any of the scripts or tools in AIM-C. This is similar to facilities in all current operating systems where the system environment is stored centrally, and all applications have access to a common set of environment variables. An example of the C shell version of the environment settings is shown in Listing 1.

```
set OS = `uname`
setenv OS_VERSION `uname -r`

if ($OS == "SunOS") then
    setenv JAVA_HOME /apps/java/Java1.4.1_01

    setenv ANSYS_HOME /usr/local/bin
    setenv ANSYS_EXE ansys60

    setenv COMPILER wsv
    setenv COMPILER_VERSION 6_2

    setenv F77 /apps/$COMPILER$COMPILER_VERSION/bin/f77
    setenv F90 /apps/$COMPILER$COMPILER_VERSION/bin/f90
```

## 2004P0022

```
setenv CC /apps/$COMPILER$COMPILER_VERSION/bin/CC
setenv INSTALL_env /usr/sbin/install

setenv GNUPLOT /home/jslee/bin/sun5.8/gnuplot_v3.7.1
if ($?xSize == 0) set xSize = 500
if ($?ySize == 0) set ySize = 500
setenv GNUPLOT_TERMINAL 'gif medium size '$xSize', '$ySize' xDDDDDD x000000
x404040 xff0000 xffa500 x66cdaa xcdb5cd xadd8e6 x0000ff xdda0dd x9500d3'
else if ($OS == "Linux") then
setenv JAVA_HOME /usr/java/j2sdk1.4.2_02

setenv ANSYS_HOME /usr/local/bin
setenv ANSYS_EXE ansys60

setenv COMPILER pgi
setenv COMPILER_VERSION rh73
setenv PLATFORM $OS'--'$OS_VERSION/$COMPILER'--'$COMPILER_VERSION

setenv F90 /apps/$COMPILER$COMPILER_VERSION/linux86/bin/pgf90
setenv F77 /apps/$COMPILER$COMPILER_VERSION/linux86/bin/f77
setenv F90 /apps/$COMPILER$COMPILER_VERSION/linux86/bin/f90
setenv CC /apps/$COMPILER$COMPILER_VERSION/linux86/bin/CC
setenv INSTALL_env /usr/bin/install

setenv GNUPLOT /usr/bin/gnuplot
setenv GNUPLOT_TERMINAL 'png medium'
else
echo "Unsupported OS: "$OS
exit 1
endif

setenv PLATFORM $OS'--'$OS_VERSION/$COMPILER'--'$COMPILER_VERSION

setenv JAXP_HOME /home/georient/jaxp-1.1
setenv XALAN_HOME /home/georient/xalan-j_2_2_D11
setenv JEP_HOME /home/georient/jep-2.23
setenv WebEE_HOME /apps/WebEE/V_1.1.0
```

```

setenv RDCS_HOME /home/adebchau/RDCS
setenv RDCS_REV 2.0
setenv RDCS_BIN bin_pioneer
setenv RDCS_PLATFORM linux2.4_gnu

setenv LOCAL_SCRATCH /local/scratch

setenv CLASSPATH $JAVA_HOME/lib/tools.jar
setenv CLASSPATH $CLASSPATH': '$JAXP_HOME/jaxp.jar
setenv CLASSPATH $CLASSPATH': '$XALAN_HOME/bin/xalan.jar
setenv CLASSPATH $CLASSPATH': '$JEP_HOME/jep-2.23.jar
setenv CLASSPATH $CLASSPATH': '$WebEE_HOME/WEB-INF/lib/WebEE.jar
setenv CLASSPATH $CLASSPATH': '$AIM_HOME/AIM-ComputationalLink/codes/java

```

### **Listing 1. Centralized Control of the Environment.**

If a new tool requires environment settings that are not currently in the AIM-C environment definition, then the new settings need to be added to the central environment set. Individual tools can access the central environment set by executing `$AIM_HOME/bin/setEnv.csh` or `%AIM_HOME%\bin\setEnv.bat`, for Unix and Windows environments, respectively.

***Requirement – All environment definition must be done through the central environment script.***

***Requirement – All tools invoked by contributed tools must be identified by full parametric path.***

Requirements basically state that there may be no explicit references to any resource with full non-parametric absolute path (including references to the developer's home directory or any system resource such as `/tmp`). Use the environment parameters defined in `setEnv` to construct a parametric path that uniquely defines the required resource.

Tools cannot rely on any environment such as an alias that is not defined in `setEnv`. For example, it is not acceptable to refer to the Ansys application as `ansys81`, since that alias may not be valid at the hosting site. Use something like `$ANSYS_HOME/bin/ansys81` instead, where the `ANSYS_HOME` environment variable is defined in `setEnv.csh`.

## **4 Module Delivery Process**

Modules typically encapsulate knowledge related to a single discipline. They must contain their documentation, and they may be implemented using a combination of languages (scripting, compiled), engineering tools such as CAD and FEM platforms. In order to minimize the number of scripting languages and promote platform neutrality, AIM-C prefers Java and compiled languages such as Fortran 90 and C++ for module implementation. Small wrapping scripts may be necessary, and the preferred scripting language is Python. Unix-specific shell scripting (`c-shell`, `korn-shell`) or text processing languages such as `awk` and `grep` are allowed, but their use is only justified if they perform a function that cannot be done with Java or Python.

**Requirement – Scripts must not implement engineering logic or data management operations.**

Each module needs to be delivered in an archive file (zip or tar) that is tagged by the version tag (Fiber\_Module\_V\_2.1.0, for example). The modules need to have a well-defined directory structure as described in the following subsections.

**4.1 Delivery Directory Structure**

Directory names need to be platform-independent, therefore, only alphanumeric characters, “dash” and “underscore” characters should be used (no spaces). Even though the Windows file system is not case sensitive, use correct case in all the file name references.

**Requirement – In the interest of platform-neutrality, spaces are not allowed in file names.**

To assure tractability of multiple versions of a tool, the top delivery directory needs to be tagged with the release tag. Additionally, a flat file organization structure can become intractable; therefore, the delivery package needs to be organized in subdirectories. Certain elements of this structure depend on the operating system version as well as compiler vendor and version. The AIM-C delivery process uses a two-level OS and platform ID to identify platform-specific directories as follows:

```
OS--versions
    compiler_vendor--compiler_version
```

Where

OS – name of the operating system as returned by the `uname` command on Unix-type systems or indicated in the content of the “OS” environment variable on Windows.

versions – names of the operating system versions as returned by `uname -r` command on Unix-type systems or the content of the “OS” environment variable on Windows. Separate the versions by the underscore character.

compiler – compiler name

compiler\_version – compiler version

For example, the files in “Windows--2000\_XP” and the subdirectory “Lahey\_Fortran--5.60h” are either binary content generated on Windows 2000 and XP using Lahey Fortran 5.60h or verification case run using that combination of OS and the Lahey Fortran 5.60h platform. An illustration of this structure is shown in Figure .

Detailed description of each directory in the delivery package is as follows:

**4.1.1 Directory build**

Automated and parametric build scripts are stored here. If creating a single `makefile` is not possible due to varying behavior of the `make` utility on different platforms, a separate directory should be created for each OS-platform pair. As a preferred alternative, if the process has been tested for a range of operating systems of the same type (Unix flavors) then a generic build procedure is delivered through a `make` file placed in the subdirectory called “Generic”. In this case, supply a `readme.txt` file that indicates the list of platforms where the process has been verified. The build process should start with a wrapper script that takes an argument indicating where the environment definition resides as indicated in the following example:

```
#!/bin/csh -f
setenv ENV_HOME $1
$ENV_HOME/setEnv.csh
make -f Makefile
rm *.o
```

In turn, the actual make instruction (make file or Ant script, for example) must be parameterized, and not have any explicit references to compiler names, or use aliases implied in the developer's environment. The compiler names should be stored in a setEnv script or batch file that the build script executes to establish the environment.

#### 4.1.2 Directory data

This is where data supporting the operation of the particular tool is stored. It is preferred not to tag these file names with version tag or date, since the version is already uniquely defined in the delivery directory name.

***Requirement – For each data module that is used in a binary format, an ASCII dump must be provided along with a process of bi-directional conversion between ASCII and binary data.***

For example, it is not acceptable to provide an Ansys database file. The script that generates the model and the database must be included. In addition to ASCII files being human readable and more repairable in case of damage, database and engineering tool vendors do not guarantee infinite backward compatibility.

***Requirement – The ultimate authority and responsibility for storing intrinsic data such as material properties, their variation, basic processing parameters, etc. is with the AIM-C design knowledge base.***

The files in the data directory may duplicate such data only to provide an interfacing mechanism between the vendor-supplied module and the design knowledge base. These files may be called input files, setup files or even databases. The AIM-C integration team will provide assistance in creating a data model in the design knowledge base and mapping it to the module-specific input files.

#### 4.1.3 Directory docs

Documentation resides in this directory. It is preferred to separate documentation intended for different audiences in sub-directories.

***Requirement – Each module must have a docs directory.***

Documents may have different restrictions on distribution, and a simple mechanism has been defined to state such constraints. The package needs to include an access filter file named `restrictions.txt` at the top level of each folder that contains any restricted data using the following format:

```
full_path filter_expression
```

where "full\_path" may refer to a file or a directory and the expression is a C-style logical expression potentially involving multiple variables that may have values of true or false. If the

filter\_expression evaluates "true", access is granted. There may be multiple lines in this file. For example if restriction.txt has two lines in a folder

```
data1.dat Northrop-Grumman && ITAR
detail\data2.dat Boeing && ITAR
```

then it gives access to data1.dat to users that are ITAR cleared and are members of the Northrop-Grumman user domain, and opens detail\data2.dat to users that are ITAR cleared and are members of the Boeing user domain.

#### **4.1.3.1 Directory docs/api**

If the tool has an API (Application Program Interface), then the document describing the calling methods is placed in this directory. The API documentation should be automatically generated from comments embedded in source code or header files similarly to the javadoc mechanism for Java.

#### **4.1.3.2 Directory docs/CCB**

The initial CCB submission document and the subsequent Change Form are included here. Since a change will result in an increment in the version tag, set the Change Form document name to the new version tag.

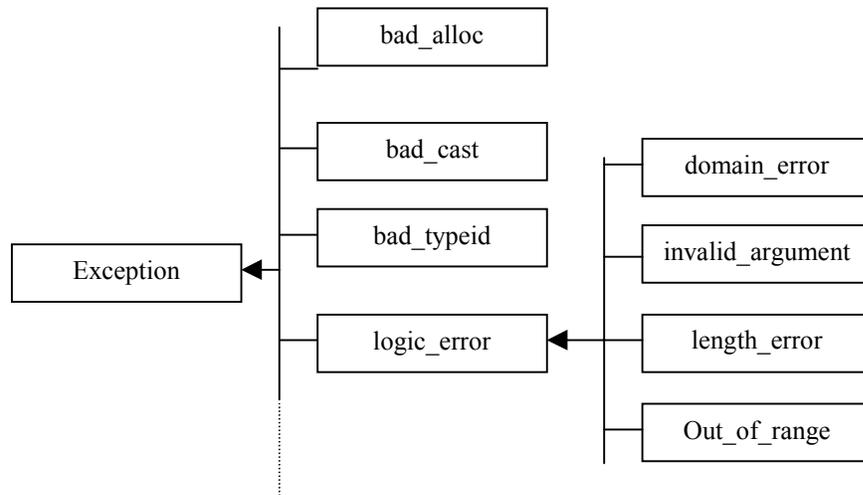
***Requirement – Each delivered software item will have a CCB directory where revision history is tracked.***

#### **4.1.3.3 Directory docs/implementation**

Implementation documentation is placed in this directory. Software implementation is documented by the following devices:

##### **4.1.3.3.1 Inheritance Tree Diagram (C++ and Java)**

Class tree diagrams are required with (hand or machine generated) inheritance or interfaces identified clearly. These diagrams can be generated using any standard modeling package (such as Booch) or, as a minimum, schematics such as that shown in Figure 4 can be supplied.



**Figure 4 Example of Class Tree Diagram**

#### 4.1.3.3.2 Member Variables and Methods (C++ and Java):

Further, a box diagram, Figure 5 identifying member variables (instance as well as static) and methods for each class is required. Both the class diagram may be generated by a CAE tool such as Rational Rose. If the number of member variables and methods allows, then the class diagram may contain these characteristics. For Java, a javadoc documentation is acceptable for inheritance tree as well as attribute and method documentation.

**Requirement – All Java code must have a complete set of javadoc tags.**

```

rdcs_DPL_Set_Container
-----
-m_total_des_pt_num : long
-m_set_list : rdcs_SingleLinkedList
-m_db_file : rdcs_File
-m_db_file_name : rdcs_ChString
-m_project_directory : rdcs_ChString
-m_project_name : rdcs_ChString
-m_design_point_db : rdcs_ChString
-----

+IsA ():rdcs_ChString
+rdcs_DPL_Set_Container (proj_dir : rdcs_ChString,proj_name : rdcs_ChString,des_pt_db : rdcs_ChString,db_name : rdcs_ChString)
-rdcs_DPL_Set_Container ():rdcs_DPL_Set_Container
+Initialize ():void
+~rdcs_DPL_Set_Container ()
+AddSet (set : rdcs_DPL_Set*):rdcs_Boolean
+DeleteSet (set_name : rdcs_ChString):rdcs_Boolean
+ReplaceSet (set_to_replace : rdcs_ChString,new_set : rdcs_DPL_Set*):rdcs_Boolean
+GetSet (name : rdcs_ChString):rdcs_DPL_Set*
+GetSetNames ():rdcs_SingleLinkedList
+ResetSetList ():void
+GetNextSet ():rdcs_DPL_Set*
+IsEndOfSets ():rdcs_Boolean
+Save (file_name : rdcs_ChString = ""):rdcs_Boolean
+GetTotalDesPoints ():long
-Read ():rdcs_Boolean
-Write ():rdcs_Boolean
  
```

**Figure 5 Documentation of member variables and methods**

#### 4.1.3.3.3 Block Diagram

In addition, for each subroutine, a block diagram describing the logic and data flow is required at a coarse grain level, identifying major logical procedural calculation steps. A detailed flow diagram at the individual statement level is discouraged.

#### 4.1.3.4 Directory docs/theory

Publications, presentations describing the engineering methodology reside here. The physics incorporated into the model will be discussed in sufficient depth that the physics-based model can be understood by a competent scientist or engineer.

The theory description should include uncertainty analysis. Characterize the uncertainty in variables in the model according to the following definitions:

- Inherent variations associated with physical system or the environment (Aleatory uncertainty)
  - Also known as variability, stochastic uncertainty
  - Manufacturing variations, loading environments
- Uncertainty due to lack of knowledge (Epistemic uncertainty)
  - Inadequate physics models
  - Information from expert opinions
- Known Errors (acknowledged)
  - Round-off errors from machine arithmetic, mesh size errors, convergence errors, error propagation algorithm
- Mistakes (unacknowledged errors)
  - Human errors, e.g. error in input/output, blunder in manufacturing

Document the results in the form of a table such as shown in Figure 6.

Think through all the uncertainties in each of the steps contained in the physics-based model: process, producibility, etc. Identify whether the uncertainties can be treated as independent or whether there is the possibility of a correlation with another uncertainty. Identify potential for human error such as the potential for manufacturing defects due to the complexity of the design or the potential for errors due to the complexity of the analysis.

Characterization and validation of physics-based models may be performed using comparison with validation data, that is, experimental data or with other physics-based models that have been characterized and validated. References should be provided to document the experimental data or other physics-based models, and the relevant data used for comparison should be included in the documentation. Validation input files and associated output files for the physics-based model are required. Comparisons should be made between the physics-based model and the validation data and assessments should be made as to (1) the accuracy of the mean value given the variabilities previously discussed, (2) the variability of the outputs characterized by higher order statistics, and (3) the range of values of the input parameters over which the physics-based model is valid.

	<p>Inherent variations associated with physical system or the environment (Aleatory uncertainty)</p> <p>Also known as variability, stochastic uncertainty</p> <p>E.G. manufacturing variations, loading environments</p>	<p>Uncertainty due to lack of knowledge (Epistemic uncertainty)</p> <p>inadequate physics models information from expert opinions.</p>	<p>Known Errors (acknowledged)</p> <p>e.g. round-off errors from machine arithmetic, mesh size errors, convergence errors, error propagation algorithm</p>	<p>Mistakes (unacknowledged errors)</p> <p>human errors e.g error in input/output, blunder in manufacturing</p>
<b>Degree of Cure</b>	batch to batch variation in rate of reaction.	Vailidity of the form of the equation; including physical basis: empirical, semi-empirical ...	Use of model outside of bounds (eg temperature range, rates). In general modules should be self-checking. Are all input parameters within predefined bounds?	DSC not calibrated; base-line choice (Need to track history of usage – at all levels. Over time this will reduce uncertainty due to this)
<b>Modulus</b>	Specimen to specimen variation; batch to batch variation.	For partially cured materials, the assumption of cure hardening, linear elastic response. For cured materials, the response under mixed mode loading.	Use of model outside of bounds (eg strain range). Approximation of straight line fit to curve.	Testing machine not calibrated. Poor specimen preparation; poor strain measurement techniques.
<b>Strength (to failure)</b>	Specimen to specimen variation; batch to batch variation.	Definition of failure; particularly for some loading cases. Initiation versus propagation of a crack.	Use of model outside of bounds (eg temperature range).	Testing machine not calibrated. Poor specimen finish, poor alignment in grips.
<b>Strain (to failure – linked to strength)</b>	Specimen to specimen variation; batch to batch variation. This value is correlated with strength and somewhat to modulus	Definition of failure; particularly for some loading cases. Initiation versus propagation of a crack.	Use of model outside of bounds (eg temperature range).	Testing machine not calibrated. Poor specimen finish, poor alignment in grips.

**Figure 6 Example of Categorization of Uncertainty for Physics-Based Model**

#### 4.1.3.5 Directory docs/user

If the tool can be run by a user, the users' instructions go here. Each module should be delivered with a user manual describing the inputs, outputs, and intended usage of the module. Further, the range of values for input parameters over which the module is expected to operate must be documented. While it is understood that extensive exercise for the validity of results for arbitrary valid combination of variables is part of the validation effort, the documentation should provide this range information for each parametric input variable.

#### 4.1.4 Directory env

Environment definition goes here. Full path to the compilers and other required resources are defined in directories named after the platform ID. An example of the environment definition script is given below:

```
setenv CC      /apps/wsv6_2/bin/f95
setenv F95     /apps/wsv6_2/bin/f95
setenv INSTALL /usr/ucb/install
```

#### 4.1.5 Directory Excel

Excel spreadsheets are stored here. Spreadsheets are valuable tools, but it takes some discipline to make them useful in the long run. There are several potentially severe problems with spreadsheets:

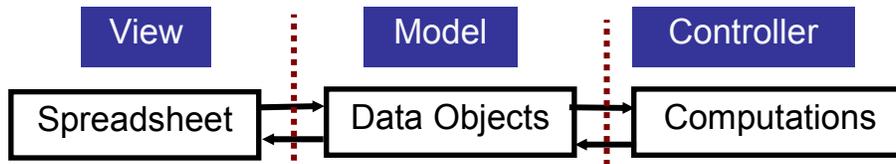
- Until Office 2004 is released, there is a chance of data corruption when an interactive user is using Excel while a batch execution of another spreadsheet is spawned by the queuing system. The root of this problem is sharing objects inherent in current Office products. A single copy of each of these shared objects resides in memory, and it is not aware of which client application is using it. This becomes an issue when a spreadsheet is utilized as a design tool in an automatic design exploration process such as those executed by RDCS.
- Due to some complex interaction between Microsoft DCOM and some queuing and load balancing systems (LSF, a commercial solution by Platform Computing, in particular), the user running the design exploration must have administrator privileges. This is usually not accepted by Information Technology (IT) organizations.
- Spreadsheets are saved as binary files. While Word documents are purely presentation elements, spreadsheets normally implement some business logic. In order to be able to discern the differences between versions, a text-based storage of the logic implementation is required. Furthermore, data should not be buried in binary spreadsheet files, either, since that would violate the requirement stated in Section 4.1.2. Treat a spreadsheet as the input and presentation software platform it is. Separate data, business logic and presentation, and enter these components in the repository separately.
- If a link to a spreadsheet is placed on the AIM-C web application, version control of the spreadsheet tool is lost. The user can download, save, and use the spreadsheet without being aware that the local copy might have been superseded.

**Requirement – Since comparison of different versions of a spreadsheet is impossible, provide a detailed and accurate description of the changes when a spreadsheet is entered in the repository. This provides a manual history of changes to the tool.**

**Requirement - Spreadsheets must not be used in automated design explorations.**

*Recommendation – Business logic should be implemented either in FORTRAN or C program compiled into a DLL.*

Commit the source code in the repository. In fact, use the spreadsheet only as a presentation and user input device, and keep the logic external, Figure 7. The externally implemented logic can easily be reused in automated design explorations.



**Figure 7 Model-View-Controller Architecture for Excel Spreadsheets**

*Recommendations – Store the input data required by the spreadsheet in .csv (Comma Separated Values) format, and provide Visual Basic functions to import the data. Commit the .csv files and the import functions to the repository.*

The biggest challenge is automatic verification of spreadsheet tools where a set of parametric inputs is provided, the spreadsheet is run, and parametric output is generated without interactively entering data in cells and initiating the calculations. One solution for this is `xlRunner`, the Visual Basic program available in the WebEE framework. The application is run in a command line as follows:

```
xlRunner spreadsheet.xls mapping.xml parametric_input parametric_output error_output
```

where

<code>spreadsheet</code>	- name of the spreadsheet file
<code>mapping.xml</code>	- name of the XML file containing input and output mappings
<code>parametric_input</code>	- name of the input file in name=value format
<code>parametric_output</code>	- name of the output file in name=value format
<code>error_output</code>	- name of the error file

A sample variable-cell mapping file is provided in Listing 2.

```
<?xml version="1.0" encoding="UTF-8" ?>

<xlr:xl-runner xmlns:xlr=http://www.boeing.rdyne.com/xlRunner
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="/xlRunner.xsd">

<description>
  Run Hatband Sizing Spreadsheet program. This file is reformatted
  to use the newest xlRunner formats. 3/11/2004. Only change from
  11/19/2003 version is that the optional "calc-to-manual" element
  was not shown in that version. - by Ray Clough.
```

```

</description>

<init>

  <xl-debug-mode>>false</xl-debug-mode>

  <!--
  Set Excel Calculation mode to Manual.  Optional: Default=true.
  The existing value will be reset at end of program.
  -->
  <calc-to-manual>>false</calc-to-manual>
  <xl-save-on-exit>>true</xl-save-on-exit>
  <result-file-overwrite>>true</result-file-overwrite>
</init>
<blocks>
  <block cardinality="1">
    <!-- this data is written directly to the spreadsheet -->
    <inputs-to-wks>
      <input wks="Data_Input" range="B3" id="Jacket_Gage" type="Single" />
      <input wks="Data_Input" range="H3" id="FSU_mixed" type="Single" />
      <input wks="Data_Input" range="I3" id="FSY_mixed" type="Single" />
    </inputs-to-wks>

    <!-- The executable functions within the Excel Workbook -->
    <exec-fcns>
      <fcfn name="exe_size_hatband_Click" location="Sheet5"/>
    </exec-fcns>

    <!-- this data is read from spreadsheet & written to the results file -->
    <result-from-wks>
      <output wks="HB_Sizing" range="L7" output-id="Wght" type="Single" />
      <output wks="HB_Sizing" range="M7" output-id="a2" type="Single" />
    </result-from-wks>
  </block>
</blocks>

</xlr:xl-runner>

```

### Listing 2 XML input to xlRunner

In this example, input variables are passed into cells of a sheet named “Data\_Input”, the VB function “exe\_size\_hatband\_Click” is executed and output values are retrieved from cells in sheet named “HB\_Sizing”. This mechanism for the current release of MS Office (Office 2002) is only recommended to record the QA (Quality Assurance) cases, and not for integration in an automated design process. Potential data corruption and queue permission problems may arise if xlRunner is used as an RDCS Math model element.

#### 4.1.6 Directory exe

Executables compiled on all platforms available to the developer are stored here in directories named after the platform ID. Considering varying interpretations of language standards, it is important to verify each tool on multiple platforms to ensure portability. Executables for each OS-platform pair are stored in an appropriately named two-level directory substructure, as described at the beginning of Section 4.1.

### 4.1.7 Directory `lib`

Load libraries and `.dll` files compiled on all platforms available to the developer are stored here in directories named after the platform identification. Considering varying interpretations of language standards, it is important to verify each tool on multiple platforms to ensure portability. Libraries for each OS-platform pair are stored in an appropriately named two-level directory substructure, as described at the beginning of Section 4.1.

### 4.1.8 Directory `QA`

The QA directory houses a set of test cases that is designed to verify that the module is functioning in accordance to the requirements. These test cases can be used to verify that the module is properly built for a new platform. As always, it is preferable that developers supply a complete test suite that exercises normal operation as well as error handling. Each QA case needs to be stored in a separate subdirectory of the main QA branch. Notice that the QA cases are not meant to be validation cases whose focus is validating the physical model, they simply check if the module works as delivered.

***Requirement - Each module must have at least one QA case.***

#### 4.1.8.1 Directory `QA\OS\platform\case_name\input`

Notice that for modules that are not platform neutral (Compiled codes), it is required that a QA directory structure is provided for each verified OS-platform pair as defined in Section 4.1. All input files and a description of the case in a `readme.txt` file are provided here. Since there is a chance that a module requires a platform independent input such as a binary file, the input directory needs to be created for each platform to facilitate automated module verification.

#### 4.1.8.2 Directory `QA\OS\platform\case_name\output`

The relevant output files used to verify the tool upon deployment are stored here.

#### 4.1.8.3 Directory `QA\OS\platform\case_name\work`

A snapshot of the actual run is given here. There may be scratch files that are not necessary for verification, but they may be useful for debugging problems.

### 4.1.9 Directory `src`

Source files are given here. For tools that utilize multiple languages, separate source files into different directories named after the languages. Each source file must start with a documentation header according to the format shown in Listing 3. Naturally, comment characters appropriate to the language need to be used.

```
// *****
// Purpose: *
// Developed By: *
// Released On: *
// Modified By: *
//      Date: *
//      Reason: *
// *****
```

### Listing 3 Documentation heading in the source code

***Requirement – All source codes for compiled and scripting languages will have a documentation heading.***

#### 4.1.10 Directory Validation

Validation data may change as the tools mature. If validation data is provided, keep it within the delivery package. There is no set format for validation data at this point. Keep the different cases in separate folders and provide tabular comparison as well as textual description. The CVS versioning system (Section 6) will not create duplicates of unchanging data, therefore there is no increasing storage needs when the same files are brought from one version to another.

An example of the directory structure in a tool delivery package for a compiled Fortran tool is shown in Figure 8.

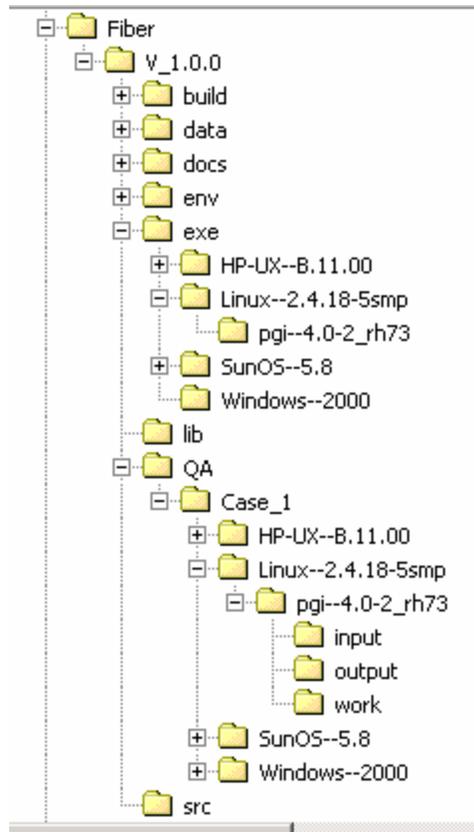


Figure 8 Tool delivery package directory structure

## 5 Template Delivery Process

### 5.1 Creating RDCS Batch File

An RDCS project is a set of directories and binary files. In order to archive a project in a way that is amenable to configuration control and version comparison, it is necessary to create an ASCII RDCS batch file. The project can be re-created from the batch file, and the batch file provides a clear and readable map of the entire project. An RDCS project is converted to a batch file as follows

```
<RDCS> export path project_name RDCS_batch_file_name
```

where

<i>&lt;RDCS&gt;</i>	- The alias or script used to invoke RDCS
<i>path</i>	- The project path. Must be followed by "/"
<i>project_name</i>	- The project name (not including the .rdcs extension)
<i>batch_file_name</i>	- The name of the RDCS batch file. The recommended extension is ".rbat"

The RDCS projects are further enhanced by parameterizing relevant information such as version of the system, working directories and design process parameters passed from either the Design

Knowledge Database or from the AIM-C system GUI. An example of such an RDCS project template is shown in Listing 4 and Listing 5.

```
BEGIN_CONTINUOUS_VARIABLE
  NAME: part_thickness
  DESCRIPTION:
  BEGIN_DOMAIN_DESCRIPTION
    PHYSICAL_BOUNDS: -8.98847e+307 8.98847e+307
  END_DOMAIN_DESCRIPTION
  BEGIN_DETERMINISTIC_DESCRIPTION
    SPEC_NAME: unknown_source
    SPEC_DESCRIPTION: none available
    MIN_NOMINAL_MAX: $part_thickness_min 0.39 $part_thickness_max
  END_DETERMINISTIC_DESCRIPTION
END_CONTINUOUS_VARIABLE
```

#### Listing 4. Templated Design Process parameters

```
BEGIN_FUNCTIONAL_MODEL
  MODEL_ID: rdcs_to_any
  RDCS_INPUT_FILE: rdcs_input

  BEGIN_SERVICE
    SERVICE: Generic:1-job-per-machine
    EXECUTABLE_NAME: $appHome'AIM-
ComputationalLink/codes/java/rdcs2any_script2.csh
    COMMAND_LINE_ARGUMENTS: $appHome
    INPUT_FILE: YES
  END_SERVICE

  BEGIN_INPUT_VARIABLES
    RDCS_NAME: part_thickness
    RDCS_NAME: pressure
    RDCS_NAME: first_hold_time
    RDCS_NAME: second_ramp_rate
    RDCS_NAME: HeatTransferCoeff
    RDCS_NAME: ToolThermalConduct
  END_INPUT_VARIABLES

  BEGIN_INPUT_FILES
    GLOBAL_NAME: $appHome'AIM-
ComputationalLink/demo2/data/'$Resin_Type'/tmp_IM7-8552.DAT'
    LOCAL_NAME: tmp_IM7-8552.DAT
    GLOBAL_NAME: $appHome'AIM-
ComputationalLink/demo2/data/'$Resin_Type'/template_file'
    LOCAL_NAME: template_file
    GLOBAL_NAME: $appHome'AIM-
ComputationalLink/demo2/data/'$Resin_Type'/tmp_ProcessModuleStart.txt'
    LOCAL_NAME: tmp_ProcessModuleStart.txt
    GLOBAL_NAME: $appHome'AIM-
ComputationalLink/demo2/data/'$Resin_Type'/tmp_Demo2_Process_Module.BCI'
    LOCAL_NAME: tmp_Dec3Demo2_Process_Module.BCI
    GLOBAL_NAME: $appHome'AIM-
ComputationalLink/demo2/data/'$Resin_Type'/tmp_Demo2_Process_Module.CYC'
    LOCAL_NAME: tmp_Dec3Demo2_Process_Module.CYC
```

```

GLOBAL_NAME: $appHome'AIM-
ComputationalLink/demo2/data/'$Resin_Type'/tmp_Invar36.dat'
LOCAL_NAME: tmp_Invar36.dat
END_INPUT_FILES
END_FUNCTIONAL_MODEL

```

### **Listing 5 Templated file locations**

In Listing 4, the lower and upper limits, `$part_thickness_min` and `$part_thickness_max` are template parameters, and they are passed from the AIM-C system user interface. The system template service looks up the appropriate numerical values, and substitutes them into an RDCS batch file that is passed to the RDCS system to generate an instance of the RDCS project. In Listing 5, the location of the data files as well as a part of the path are parametric. The `$appHome` system variable is constructed by the AIM-C system from its URL which carries the version tag; therefore, it points to the application home for the version of AIM-C being used. By referencing `$appHome` in the path, it is assured that the appropriate data files are retrieved. Notice that `$appHome` is also passed to the script executing the functional so that it can call the appropriate environment definition, `setEnv.csh` (see Section 0). The other template variable in this example is `$Resin_Type`. This variable is passed from the GUI, and it is used to reference data files specific to a resin type. The process of building the templated RDCS batch consists of the following three steps:

- Use the RDCS GUI to build and debug a new project
- Export the project into a batch file
- Replace all references to full path names with AIM-C system parameters or variables that are specific to a tool.

## **5.2 Configuration Control of Tools in Functional Models**

When the design template is delivered, all the reusable modules (fiber, resin, etc.) must be in configuration control, therefore the executables will carry the appropriate version tag. Other components developed specifically for the template will be delivered in whole with the template in a separate directory of their own. For tools that are not under configuration control of their own, a release package should be created within the directory structure of the template.

In order to assure easy porting of templates between different environments, all configuration settings must be centralized. In other words, no references to users home directories, paths to tools or other environment settings should be in any of the scripts or tools in AIM-C.

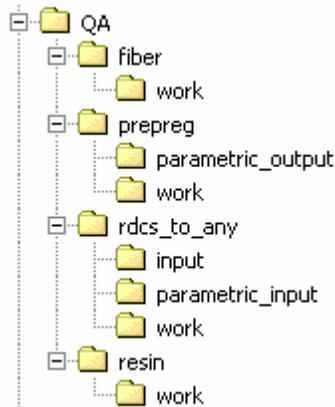
## **5.3 Quality Assurance - Verification of the Math Model Outside of RDCS**

This delivery item ensures that the design template is verifiable without running it in RDCS. The main purpose is facilitating deployment of the template in an environment other than the template developers' operating environment. An additional benefit is that all the functionality of the template is recorded, facilitating stating the template as a single script or transferring it to an alternate design framework. The process is as follows:

- Create a directory for each functional model.
- Create a work directory where the functional model will run.

- If the functional model has parametric input, create a directory called “parametric\_input”, and place a copy of the parametric input file there.
- If the functional model has independent input, create a directory called “input”, and place a copy of all the independent input files there. Alternatively, provide a script that stages the independent files from a known location stated as a relative path.

An example for the QA directory structure is shown in Figure 9.



**Figure 9 Directory Structure for Math Model Verification.**

Finally, write a file called “run.template” in the directory for each functional model. This file serves as a template for running the functional model stand-alone by copying lines from it as illustrated by the following example for the “prepreg” functional model:

```

# Copy independent input file

cp /apps/AIM-C/V_0.0.4/AIM-ComputationalLink/demo4/data/PrepregPropertiesList.txt .

# Copy files from link between functional models

cp ../../rdcs_to_any/work/Prepreg-IM7_977-3_Ver-Nov7-2001_User_Input.txt .
cp ../../rdcs_to_any/work/Prepreg-IM7_977-3_Ver-Nov7-2001.txt .

cp ../../fiber/work/FIBMOD_SETUP.out .

cp ../../resin/work/Resin_977_Ver-25Nov-2001.out .

# Execute functional model

/apps/AIM-C/V_0.0.4/AIM-ComputationalLink/codes/Prepreg_module5/Prepreg

```

## 5.4 Directory Structure

If a template has a utility application whose applicability is restricted to the template, then a complete module delivery directory structure needs to be created within the template directory. It is often possible to abstract the utility and generalize its applicability. In this case, it may be delivered either as a Module or as a Utility.

### 5.4.1 Directory `docs`

Templates may have access restrictions that can be handled through the `restrictions.txt` file described in Section 4.1.3. For templates, the `docs` directory has some of the elements defined for Modules in Section 4.1.3, specifically the `CCB`, `theory` and `validation` directories. Since math models are aggregates of multiple physics modules, an assessment of their ability to predict variability needs to be performed and documented in the `theory` directory. If the template has utilities with exclusive relevance to it, the `API` or `user` directories need to reside in the directory created for the utility within the template directory.

***Requirement – Each template shall have a CCB and a theory documentation directory.***

### 5.4.2 Directory `QA-Math_Model`

Results of the Math Model verification as described in Section 5.3.

### 5.4.3 Directory `QA-Design_Process`

When running design processes in an RDCS project, output files are generated in the Output directory within the RDCS project directory. Provide copies of these files here so that correct operation of the design processes can be verified.

### 5.4.4 Directory `rdcs`

The RDCS batch file generated from the RDCS project resides here. Although there is no requirement for file extension on the RDCS batch file, use of `.rbat` extension is recommended.

Additionally, each tool that is used only in a particular template needs to have its own delivery directory structure that conforms to the module delivery requirements specified in Section 4. The number of such tools should be kept minimal. Instead, consider generalizing the tool and placing it in the global Utilities repository.

## 6 The Configuration Control Process

### 6.1 CVS – Concurrent Versioning System Overview

Client/server based CVS enables developers scattered by geography or slow modems to function as a single team. The version history is stored on a single central server and the client machines have a copy of all the files that the developers are working on. Therefore, the network between the client and the server must be up to perform CVS operations (such as check-ins or updates) but need not be up to edit or manipulate the current versions of the files. Clients can perform all the same operations that are available locally.

CVS' basic version control functionality maintains a history of all changes made to each directory tree it manages, operating on entire directory trees, not just single files.

CVS supports branches allowing several lines of development to occur in parallel and providing mechanisms for merging branches back together when desired.

CVS can tag the state of the directory tree at a given point, recreate that state and display the differences between tags or revisions in the standard diff formats.

CVS has Unreserved Checkouts allowing more than one developer to work on the same files at the same time. Using CVS in this mode is not recommended in AIM-C.

CVS provides a flexible modules database that provides a symbolic mapping of names to components of a larger software distribution. It applies names to collections of directories and files. A single command can manipulate the entire collection.

CVS provides reliable repository access to remote hosts using Internet protocols, facilitating collaboration with distant employees and contractors.

All CVS operations can be performed over the network. A developer on a remote host can check out a local copy of the sources, make changes, update her local copy with changes made by others and commit her changes back into the repository.

Remote operation is efficient, transmitting only those files that have changed. When appropriate, CVS transmits patches to files and verifies the results rather than sending entire files. CVS can compress the text it transmits.

Remote operation is reliable. CVS holds no internal locks while waiting for communications to complete so network troubles will not disrupt others' access to the repository. It uses reliable transport mechanisms, not NFS, making it well adapted for use over wide-area networks.

Remote operation is authenticated. CVS can use the industry standard Kerberos protocols to verify the identity of the remote user. Kerberos is much more secure than the source-address authentication provided by the ordinary rlogin and rsh protocols. CVS can also work with ssh, a secure replacement for rsh, or use straight password authentication.

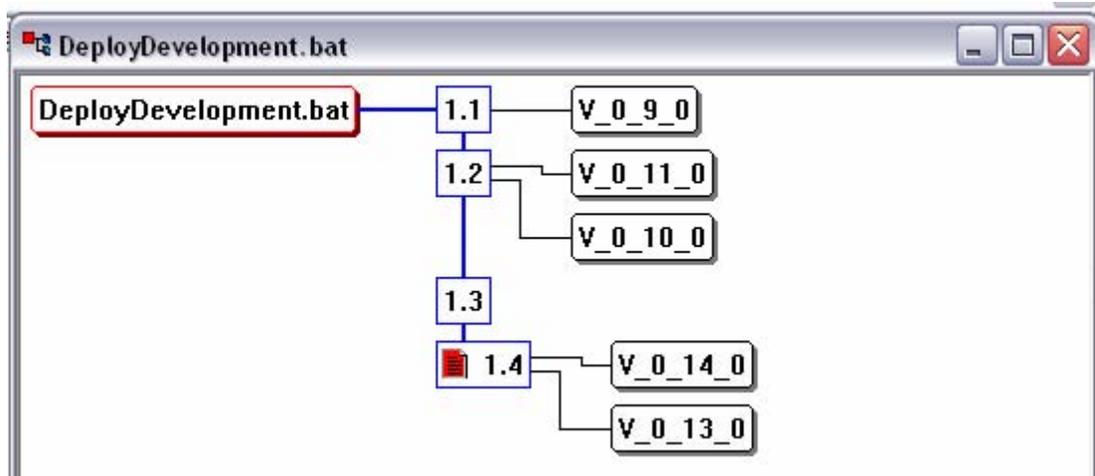
Remote operation supports portable computers. While a developer's portable computer is connected to the network, she can update her working copy of the source code. While disconnected, she can develop her working copy. The next time she connects to the network, she can commit her changes.

More information is available at <http://www.cvshome.org>.

## **6.2 Tagging the Repository**

It is recommended that at the end of each workday all files are entered in the repository. Since released versions are specifically tagged with the version tag, there is no danger of other developers inadvertently accessing an unreleased resource. Developers should use only released versions of files for which they are not directly responsible.

Since CVS does not allow the “dot” character in version tags, the versions tag used in CVS uses “underscore” characters to represent the AIM-C version tag. In the snapshot shown in Figure 10, the DeployDevelopment.bat file is shown to have gone through 4 revisions. Version 1.3 was an intermediate configuration not worth tagging, but 1.4 is part of both V\_0.13.0 and V\_0.14.0 releases.



**Figure 10 Application of the Version Tag in CVS**

While CVS has many features and development practices to perform multi-branch development, the following actions seem adequate to for simple version control:

*Import* – The initial act of entering a module in the repository

*Commit* – Uploading of files in the repository (preferably daily)

*Update* – Downloading files from the repository (preferably daily)

*Tag* – Attach a version tag to a consistent set of files that comprise a release. It requires consensus among the configuration control board members. It is not to be confused with the version number automatically assigned to the daily commits.

*Checkout* – Creating a new image of a tagged version to support delivery of the product.

## 7 The Change Control Process

The AIM-C modules, templates, and system software have reached a maturity level and volume where configuration, versioning, and change control are required to maintain schedule, assess and control ripple effects due to changes in components of methodology or their software implementation. The basic approach to the Change Control Process in Reference [2] has been adopted and modified to suit engineering methodology and software. The goal is to institute a simple to follow processes that does not put undue burden on methodology and tool developers, but one that is effective at allocating the right resources to the right work package.

A Configuration Control Board (CCB) with representatives of each engineering discipline and the integration team has been established to serve as forum where change proposals are analyzed and a course of action is agreed. The CCB has several objectives:

- Establish a baseline methodology and software implementation.
- Triage changes to optimize resources and schedule. Only changes supported by consensus will be implemented.
- Allow all parties that would be affected by changes to assess the impact of proposed changes.
- Document and record a change history for each module, template or other element of the methodology.

The configuration control board has to decide each proposed change by considering the classic trade-off triangle: schedule, cost, and features with the added dimensions of engineering accuracy, and long-term architectural viability. A scorecard will be developed to quantify and document the multidimensional decision process the CCB practices.

The steps of development followed on a work article under change controls are as follows:

- Submit change request
- Implement the change
- Go through the Software Release Process of appropriate length to assure quality
- Submit the completed work article to the CCB
- Execute the steps of the Detailed Change Control Workflow (Section 7.2.1.1) to release the new version of the package

The initial steps undertaken by the CCB are focused on entering all tools (modules and templates) in version control by modifying and extending the configuration control processes practiced by the integration team to general work articles.

### **7.1 The Change Request**

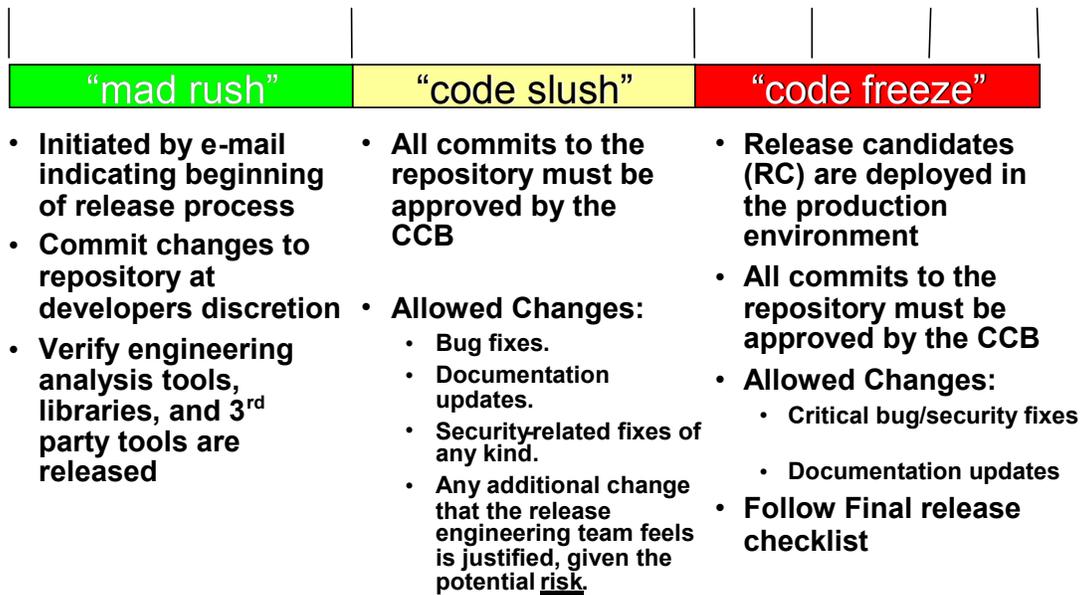
Before a change on the Phase I AIM-C product is worked, a simple Change Request document is submitted. The document lists the following items:

- Overview – Description of the changes
- Benefits - How is the change benefiting AIM-C
- Impact – Assessment of other modules impacted
- Approach – How is the change to be implemented
- Resource needs – Schedule, budget, etc.
- Public Interface – How is the interface to the module changed
- Impact Statement – What are the templates and modules known to be impacted by the change.

Upon submission, the CCB is expected to review and dispose the request within one session, unless assessment of impact requires more time.

### **7.2 The Software Release Process (SRP)**

After the Change Request is approved, development starts on the work article. Before a unit of methodology implemented as a software package is submitted to the CCB, it needs to go through its own internal release process. Three stages of the release process are “mad rush”, “code slush” and “code freeze”, as depicted in Figure 11. Committing changes in the repository is slowed down at each stage, effectively instituting a configuration control step whose scope is limited to the particular package. The length of each phase depends on the complexity of the delivery item. Modules may take a short time (4-6 days) to complete the SRP, while larger units such as the AIM-C system itself will take about 6 weeks.



**Figure 11 Software Release Process**

### 7.2.1.1 Detailed Change Control Workflow

Once a package passes through its own release process, it is submitted to the CCB. The

- 1 - Developers create an initial CCB submission package of a module or template.
- 2 - Formal requirements are iterated/reviewed, and the package is accepted
- 3 - Version V\_1.0.0 is assigned to the package
- 4 - A member of the CCB or a representative of the developer(s) commits the package in CVS repository. Access permissions need to be set up for each CCB item in the repository to read/write by CCB members and members of the package developer team. The basic idea is that the developer team uses the same repository as CCB, but a version tag is only assigned with CCB approval.
  - 4.1 - Import package from working directory to the repository
  - 4.2 - Temporarily save a copy of package in the working directory
  - 4.3 - Checkout module from the repository to the working directory. This step results in a version controlled package in the working directory. Note: when multiple developers collaborate, there will be multiple working directories. However, only content committed in the repository is relevant for the AIM-C project, therefore the process of committing and updating work regularly needs to be routine.
- 5 - The repository is tagged with the current version tag.
- 6 - If needed, package developers continue to work on the capabilities/features. They commit their changes daily as long as the package has not entered formal release process leading up to CCB submission. If the package is in release process, then commits are governed by the release process rules.
- 7 - At the end of the release process the package is submitted to CCB
- 8 - Upon approval, a new version tag is assigned by the CCB, a deploy script is generated and committed to the repository. The CSV repository is tagged with the new version tag.
- 9 - Using the deployment script, the current package is deployed from the work area to the Release area for general use. Make sure that the new version of the tool is deployed into a

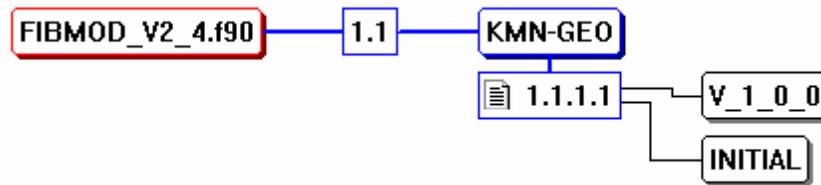
separate directory named after the version tag. Some templates may keep using older versions of modules until the templates are updated.

10 - Release area permissions for the released package are set to read only

11 - The deployment script is removed from the work area and the repository to discourage re-release of the same version of the package. For emergencies, the deployment script is still available in the "attic" of the repository.

12 - GOTO 6

A snapshot of one of the files in the CVS repository for the Fiber Module is shown in Figure 12. The graph shows internally assigned version numbers 1.1 and 1.1.1.1, which are used by CVS to track daily commits in the repository. The tag V\_1\_0\_0 is of relevance, since this version tag assigned by the Change Control Board is the handle used to retrieve a consistent set of files associated with V\_1.0.0. Different files are likely to have been committed a different number of times, thereby having different internally assigned version numbers. The version tag associates a consistent set of files that was most current at the time of tagging the repository.



**Figure 12 Versions in CVS Repository**

#### References:

[1] G. Havskjold: Robust Design Computational System AFRL-ML-WP-TR-2000-4093 (2000)

[2] S. McConnell: Rapid Development; Microsoft Press (1996)