The TEXspecTool for Computer Aided Software Engineering

Stephen E. Oliver
Whileshell Labs, Pinawa, Manitoba, Canada R0E1A0
olivers@aecl.ca

Introduction

This paper reviews the development of the TEXspec tool, which assists in the development and documentation of quality assured software in a regulated environment. The tool can assist in the development of a broad range of software, but targets the development of software that implements mathematical models. The original application relates to the development of models of a repository for Canada's high level nuclear waste, but is not limited to this use. TEXspec is particularly useful when documenting models and associated programs which rely on mathematical notations to communicate the intent of the software.

Problem Definition

Canada has developed computer programs to model a deep geologic repository for used nuclear fuel [3] [2]. Regulators require that these programs be of demonstrably high quality to support licence applications.

In 1999, the Canadian Standards Association (CSA) adopted standard N286.7 [6] for the development of nuclear safety related computer programs, a scope that includes the AECL models. While the software development process used to date had been considered robust, it required refinement in order to achieve compliance with this standard.

The TEXspec project seeks to address the issue of compliance with CSA N286.7. The tool supports a compliant software development procedure, while imposing a minimum of additional overhead. While optimised to meet requirements associated with the modeling nuclear fuel waste, it is hoped that TEXspec will find more common usage.

Several commercial Computer Aided Software Engineering (CASE) tools will support a robust software development methodology, but none provide support for the mathematical notations that are common in scientific models. TEXspec provides extensive support for this notation.

Software Development Methodology. Although Object Oriented (OO) analysis and design is appropriate for documenting many software applications, there are still applications for procedure/flow based software. In particular, some models which are basically linear in structure are best described using structured (non-OO) methodologies.

Many scientific models have, to date, been described using a modified Yourdon/DeMarco methodology [5, 11]. Although OO methods would perhaps be more appropriate for some models, priority is given to the more common Yourdon/DeMarco analysis methodology. Products associated with this methodology are:

- data flow diagrams (DFDs),
- process descriptions (mini-specs),
- structure charts,
- subprogram design descriptions, and
- data dictionary listings.

DFDs and mini-specs comprise the requirements specification, while structure charts and subprogram design descriptions document the design. Data dictionary listings may be separated into requirements and design, or combined into a single product.

Requirements Specification. Figure 1 illustrates the main concepts of data flow diagrams. Diagram 0 shows the input and output 'flows' to/from a single 'process'. This high level abstraction is intended to allow the reader to identify the functions of the complete system. Process 1 is broken into components in Diagram 1. The diagrams may be thought of as a hierarchy, with higher level diagrams having shorter process numbers (i.e. Diagram 1.2.3 is 'higher' than Diagram 1.2.3.4) The numbering convention of the diagrams and the processes allows the decomposition to be clearly seen. Once processes are decomposed to a point where they can be clearly specified in a short textual description, they appear on a DFD with a double circle, as Process 1.4 illustrates. Such 'atomic' processes are associated with a mini-spec, rather than a lower level diagram.

Data flows, like processes, can be broken up into constituent parts on lower diagrams. In figure 1, for example, 'Implements' on Diagram 0 becomes 'Measuring Cup', 'Bowl', 'Oven', and 'Pan' on Diagram 1. These flows are associated with multiple processes on Diagram 1, so they must be shown individually.

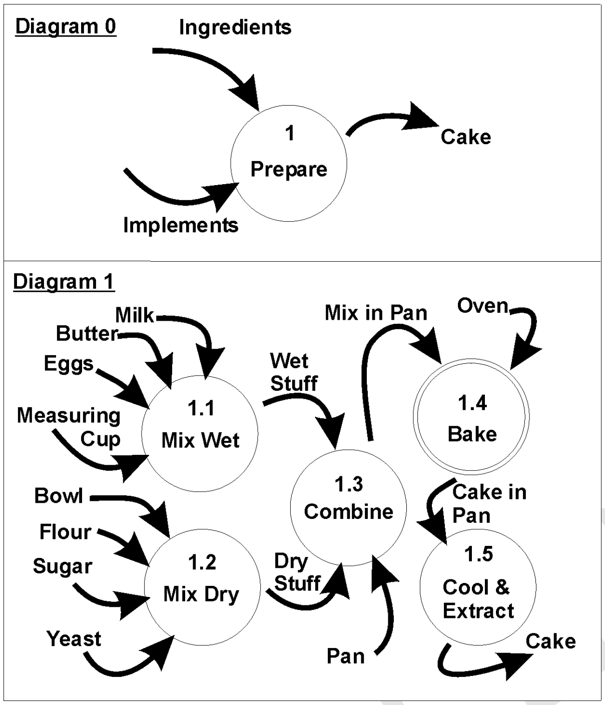


Figure 1: Example Data Flow Diagrams (DFDs)

Mini-specs for each atomic process repeat some of the information on the DFD, and also detail the requirements for a low level process in any manner deemed suitable by the author. For scientific codes, mini-specs often make extensive use of diagrams and mathematical notations.

Design Specification. Figure 2 illustrates the main concepts of structure charts. The boxes represent 'subprograms' to be composed in a procedural programming language such as FORTRAN. The chart is intended to illustrate the nature of the interface between subprograms. The lines between the subprograms indicate a 'calling' relationship, with the subprogram which is closer to the top of the structure chart invoking the lower. The transfer of data at these interfaces is also shown. Data can be passed from one subprogram to another via an argument list (shown along the connecting line), or through common storage that can be accessed from multiple subprograms (shown inside the subprogram box). The interface variables can be input, output, or both, as denoted by arrowheads next to the variable name.

Structure charts do not have a hierarchical organization paralleling DFDs. However, a large structure chart may span many pages using off-page connectors.

Each subprogram must itself be documented. Subprogram design descriptions repeat some of the information on the structure chart, and document the algorithm and design details of the subprogram. This may include material common with mini-specs, as the design reflects the requirements. In particular, many of the mathematical equations are referenced in both places.

Consistency Between Products. Experience at AECL has shown that lack of consistency between software products has been a major source of software defects [7].

Commercial CASE tools have helped to reduce this inconsistency, but these tools all have difficulty in one or more critical areas:

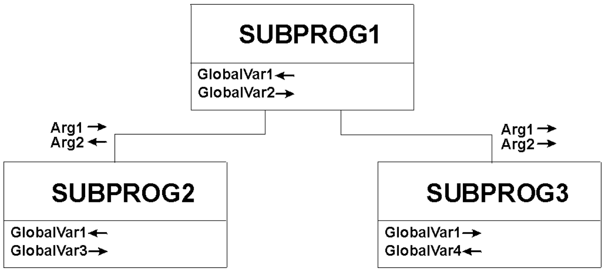


Figure 2: Example Structure Chart

- Lack of support for scientific and mathematical notations. The nature of scientific software demands that mathematical notations (e.g., $A_i(t) = \int_0^t \left[F_i^{IN}(\tau)\right] d\tau$) be permitted in specifications.
- Insufficient accountability. The principle of ownership and accountability for products is not strictly enforced. While a record of who updated products is often kept, the process control is typically inadequate.
- Assembling large products from smaller components is not adequately supported. Many defects originate as transcription errors between products. Mathematical equations are particularly susceptible.
- Insufficient consistency checking between products.

The TEXspec Solution

TEXspec takes advantage of the plain text nature of LATEX input to permit processing and tracking of shared components. The main TEXspec processing is performed by modules which have been implemented in PERL [10], as indicated in figure 3. A graphical user interface (GUI) captures interactions with the user. Most of this interaction consists of displaying and manipulating 'component' files, which form the inputs for the TEXspec scripts that select components and assemble them into products. These products are primarily LATEX or Noweb [8] input files, which can be post-processed to produce output suitable for viewing, printing, or compiling.

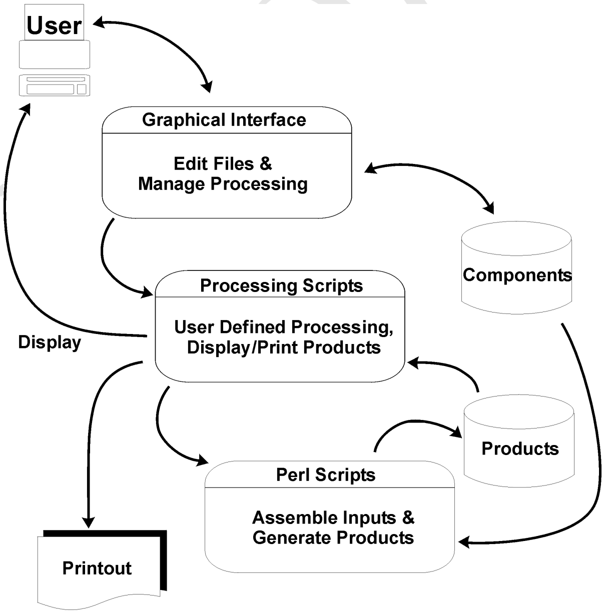


Figure 3: TEXspec Architecture

While the GUI is a convenient way to construct components and initiate processing, it can be bypassed if required. The components can be generated by any means that can generate an plain text output file, including a text editor. More importantly, the processing can be controlled by any means that can initiate a process, with no requirement for interaction with a GUI. When processing many components, or when a log of processing is required, this 'batch' style processing is a useful alternative.

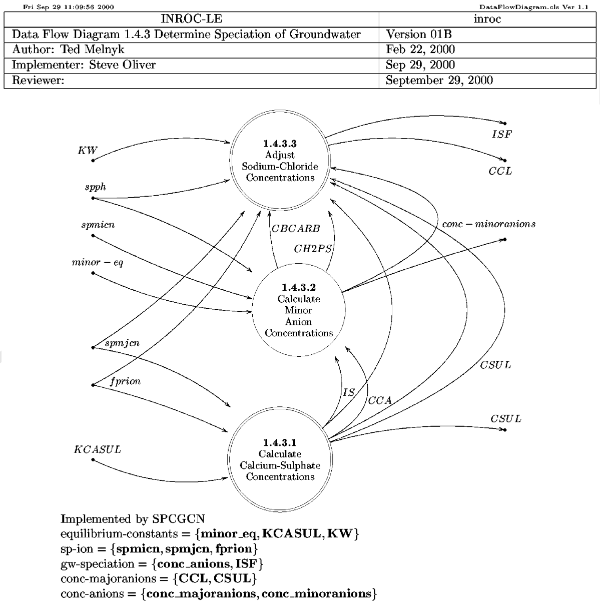
Neither the TEXspec scripts, nor the GUI can display or print the products. Figure 3 indicates that an intermediate script, which is intended to be edited by the user, initiates TEXspec to produce the product files, then controls post-processing as appropriate. This flexibility allows the user to integrate TEXspec into existing procedures. For example, if a static code analyzer such as Floppy [1] is in use, it can be run automatically on code as it is generated. Interaction with a version control system might be desired, or the user may even wish to compile code as it is generated. Alternatively, processing that is not needed can be removed, such as removing documentation generation (including LATEX processing) until the code is stable.

In order to support sharing of equations and data definitions, while tracking ownership and responsibility for content, TEXspec supports a fine granularity of components. Each TEXspec component is tracked independently by placing each in a unique file which is mapped by the file name to the name of the component, and by the file name 'extension' (in the tradition of MS-DOS or CP/M) to the type of component.

TEXspec components, with associated file name extensions, are:

- Requirements Data Dictionary entries (**.rdd**),
- Design Data Dictionary entries (**.ddd**),
- Equations (**.teq**),
- Data Flow Diagrams (**.dfd**),
- Mini-Specs (**.ms**),
- Structure Charts (**.sc**),
- Subprogram Design Descriptions (**.ds**), and
- Manuals (**.tex**).

Data Flow Diagrams. DFDs such as figure 4 are produced using the xy-pic package [9] under LATEX.



Fri Sep 29 11:09:56 2000 DataFlowDiagram.cls Ver 1.1

INROC-LE	inroc
Data Flow Diagram 1.4.3 Determine Speciation of Groundwater	Version 01B
Author: Ted Melnyk	Feb 22, 2000
Implementer: Steve Oliver	Sep 29, 2000
Reviewer:	September 29, 2000

KW
spph
spmicn
minor − eq
spmjcn
fprion
KCASUL
1.4.3.3 Adjust Sodium-Chloride Concentrations
1.4.3.2 Calculate Minor Anion Concentrations
1.4.3.1 Calculate Calcium-Sulphate Concentrations
CBCARB
CH2PS
IS
CCA
ISF
CCL
conc − minoranions
CSUL
CSUL

Implemented by SPCGCN
equilibrium-constants = {**minor_eq, KCASUL, KW**}
sp-ion = {**spmicn, spmjcn, fprion**}
gw-speciation = {**conc_anions, ISF**}
conc-majoranions = {**CCL, CSUL**}
conc-anions = {**conc_majoranions, conc_minoranions**}

Figure 4: TEXspec Data Flow Diagram (DFD)

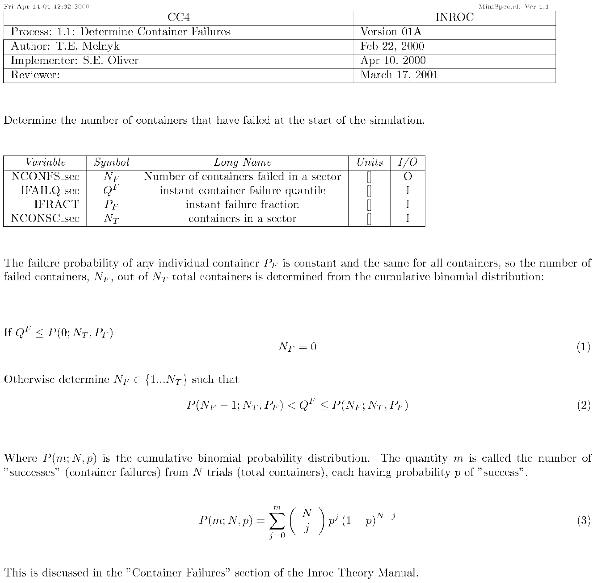
TEXspec shows the details of composite data flow decomposition explicitly on the DFD. In figure 4, for example, flow 'conc-majoranions' (which would appear on Diagram 1.4.3) is shown as constituent components 'CCL' and 'CSUL'. If a composite flow contains components which do not appear on the current diagram, they are shown in a regular font, while components that do appear on the current diagram are shown in a bold font.

Consistency between DFDs is monitored by TEXspec. A warning messages is generated for any inconsistency between a DFD and it's parent.

Labels can be shown with mathematical notation, rather than the plain text shown. Switching from plain text to mathematical labels is simple, since flows are taken from the requirements data dictionary (file **name.rdd**), which typically contains both a mathematical and a plain text label. Although this is an interesting capability, there has been little enthusiasm among users to take advantage of it.

The format of the header in figure 4 is common to all TEXspec products, detailing the project, the responsible author, implementer, and reviewer, along with an indication of the genealogy of the product (in very small type), which can be used to trace back the source of any defects.

Mini-Specs. Atomic processes are not broken down into lower level DFDs, but are further specified using a mini-spec. This document is intended to be flexible in format, permitting the author freedom to communicate the intent of the process in whatever manner is most effective.



Fri Apr 14 01:42:32 2000 MiniSpec.cls Ver 1.1

CC4	INROC
Process: 1.1: Determine Container Failures	Version 01A
Author: T.E. Melnyk	Feb 22, 2000
Implementer: S.E. Oliver	Apr 10, 2000
Reviewer:	March 17, 2001

Determine the number of containers that have failed at the start of the simulation.

Variable	*Symbol*	*Long Name*	*Units*	*I/O*
NCONFS_sec	N_F	Number of containers failed in a sector	[]	O
IFAILQ_sec	Q^F	instant container failure quantile	[]	I
IFRACT	P_F	instant failure fraction	[]	I
NCONSC_sec	N_T	containers in a sector	[]	I

The failure probability of any individual container P_F is constant and the same for all containers, so the number of failed containers, N_F, out of N_T total containers is determined from the cumulative binomial distribution:

If $Q^F \leq P(0; N_T, P_F)$

$$N_F = 0 \tag{1}$$

Otherwise determine $N_F \in \{1...N_T\}$ such that

$$P(N_F - 1; N_T, P_F) < Q^F \leq P(N_F; N_T, P_F) \tag{2}$$

Where $P(m; N, p)$ is the cumulative binomial probability distribution. The quantity m is called the number of "successes" (container failures) from N trials (total containers), each having probability p of "success".

$$P(m; N, p) = \sum_{j=0}^{m} \left(\begin{array}{c} N \\ j \end{array} \right) p^j \, (1-p)^{N-j} \tag{3}$$

This is discussed in the "Container Failures" section of the Inroc Theory Manual.

Figure 5: TEXspec Mini-Spec (MS)

The standard TEXspec header is generated for each mini-spec, as shown in figure 5. The author must explicitly state input and output flows, which are presented in tabular format and verified for consistency with the DFD.

Equations appearing in mini-specs are often referenced in other documents. Authors are encouraged, but not required, to place each equation in a separate file (**name.rdd**), and reference that file from within the mini-spec. The equation can then be reused in subprogram design descriptions or manuals. At AECL, a commercial package is used to create equations that can be saved in LATEX format which includes information encoded as LATEX comments which permits reuse by word processors. By keeping an equation in a single file, available for reuse, transcription errors are reduced.

Structure Charts. TEXspec structure charts such as figure 6 are produced using the xy-pic package [9] under LATEX. Subprograms can be grouped using colour coded backgrounds. In figure 6, subprograms 'SOURCE', 'ZAPINT', and 'REPFUN' are grouped with a yellow background indicating that they are library routines, not part of the software being documented.

Input and output variables are also colour coded: green for input, red for output, and blue for both. This applies to both the arguments to a subprogram and to the common storage variables.

FORTRAN groups common storage variables into named blocks, which are indicated to the left of each variable name. Blocks are sorted alphabetically, and variables are sorted alphabetically within a block.

For subprograms that are functions, rather than subroutines, an additional output variable is provided in the name of the subprogram itself. In this case the name of the subprogram appears in red.

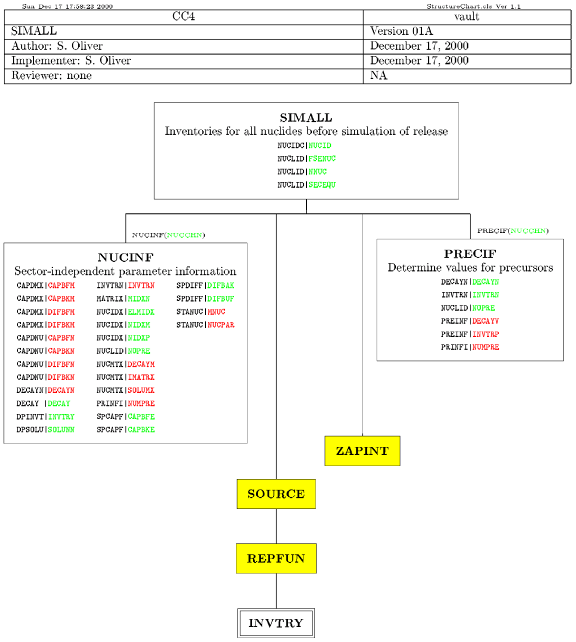


Figure 6: TEXspec Structure Chart

Most of the information on the structure chart is extracted from the subprogram design description (**subprogram.ds**) file for each subprogram on the chart. Presentation details are contained in a structure chart file (**name.sc**). The structure chart file lists the subprograms to be placed at specified locations. It also indicates the calling relationship between the subprograms and the location of the connecting lines. TEXspec generates a warning message if the specified calling relationship is not consistent with the FORTRAN code.

The description of the subprogram, and the argument list, are extracted from the subprogram design description file, and can be quite long. The structure chart may specify a maximum line length for these elements, and TEXspec inserts line breaks appropriately.

The double box 'INVTRY' at the bottom of figure 6 is an off-page connector to another structure chart. This chart is referenced by an off-page connector 'SIMALL' on another chart.

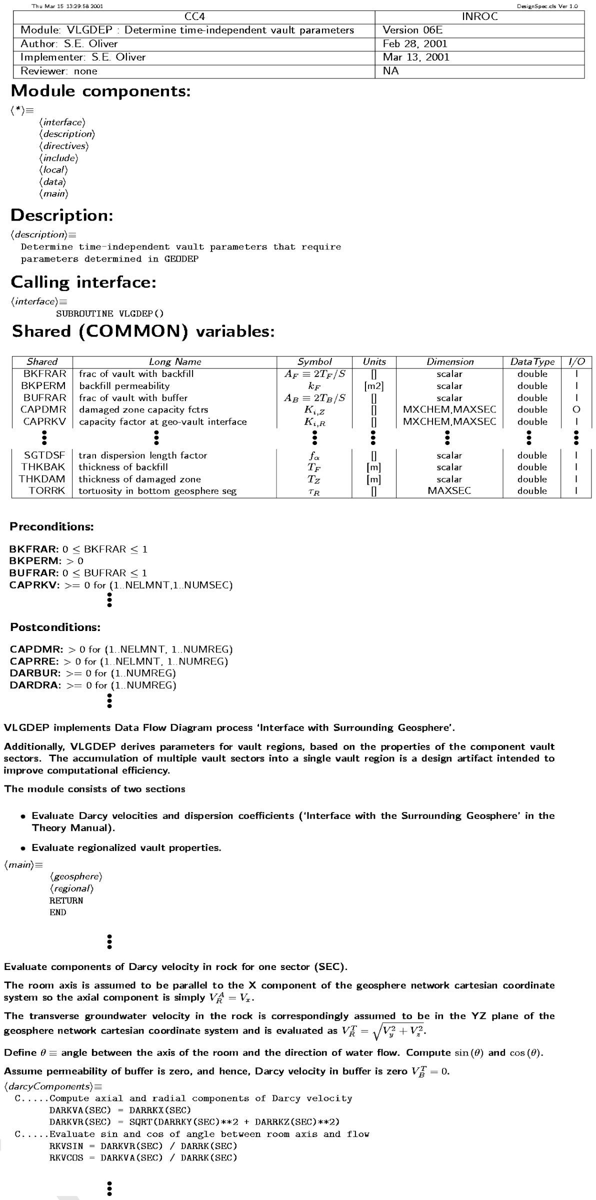
Subprogram Design Descriptions. TEXspec supports literate programming techniques [4] via use of the Noweb [8] package. Some preprocessing and postprocessing is required to achieve the desired products, but Noweb users will be immediately familiar with the format.

Subprogram design descriptions can be long documents, but an abrieviated product is shown in figure 7. The TEXspec header is followed by a Noweb-style list of code blocks that comprise the subprogram.

Code blocks to declare variables are replaced by a tabular form which contains additional data dictionary information. Of particular interest is the 'Symbol', which is the mathematical notation for a variable. This allows variables to be traced through equations, making the relationship between code and requirements much easier to follow. Also specified is the input/output status of each variable, which TEXspec validates against the contents of the code blocks.

Several tables may be generated. One each for subprogram calling arguments, common storage variables, local storage variables, and constants. After each table, the design may optionally specify preconditions and postconditions for the tabulated variables. Specifying valid ranges for data has proven to make testing much more accurate, and the process of specifying those ranges has identified many defects. Specifying preconditions and postconditions early in the design process is an effective and inexpensive quality control device.

Following the tables are the code blocks, each including commentary in LATEX format which may feature equations shared with mini-specs or other documents. Sharing equations ensures consistency. Consistency between subprogram design descriptions and compilable code is guaranteed, since Noweb extracts the code



Thu Mar 15 13:29:58 2001 DesignSpec.cls Ver 1.0

CC4	INROC
Module: VLGDEP : Determine time-independent vault parameters	Version 06E
Author: S.E. Oliver	Feb 28, 2001
Implementer: S.E. Oliver	Mar 13, 2001
Reviewer: none	NA

Module components:

```
⟨*⟩≡
      ⟨interface⟩
      ⟨description⟩
      ⟨directives⟩
      ⟨include⟩
      ⟨local⟩
      ⟨data⟩
      ⟨main⟩
```

Description:

```
⟨description⟩≡
  Determine time-independent vault parameters that require
  parameters determined in GEODEP
```

Calling interface:

```
⟨interface⟩≡
        SUBROUTINE VLGDEP()
```

Shared (COMMON) variables:

Shared	*Long Name*	*Symbol*	*Units*	*Dimension*	*DataType*	*I/O*
BKFRAR	frac of vault with backfill	$A_F \equiv 2T_F/S$	[]	scalar	double	I
BKPERM	backfill permeability	k_F	[m2]	scalar	double	I
BUFRAR	frac of vault with buffer	$A_B \equiv 2T_B/S$	[]	scalar	double	I
CAPDMR	damaged zone capacity fctrs	$K_{i,Z}$	[]	MXCHEM,MAXSEC	double	O
CAPRKV	capacity factor at geo-vault interface	$K_{i,R}$	[]	MXCHEM,MAXSEC	double	I
⋮	⋮	⋮	⋮	⋮	⋮	⋮
SGTDSF	tran dispersion length factor	f_α	[]	scalar	double	I
THKBAK	thickness of backfill	T_F	[m]	scalar	double	I
THKDAM	thickness of damaged zone	T_Z	[m]	scalar	double	I
TORRK	tortuosity in bottom geosphere seg	τ_R	[]	MAXSEC	double	I

Preconditions:

BKFRAR: $0 \leq$ BKFRAR ≤ 1
BKPERM: > 0
BUFRAR: $0 \leq$ BUFRAR ≤ 1
CAPRKV: >= 0 for (1..NELMNT,1..NUMSEC)
⋮

Postconditions:

CAPDMR: $>$ 0 for (1..NELMNT, 1..NUMREG)
CAPRRE: $>$ 0 for (1..NELMNT, 1..NUMREG)
DARBUR: >= 0 for (1..NUMREG)
DARDRA: >= 0 for (1..NUMREG)
⋮

VLGDEP implements Data Flow Diagram process 'Interface with Surrounding Geosphere'.

Additionally, VLGDEP derives parameters for vault regions, based on the properties of the component vault sectors. The accumulation of multiple vault sectors into a single vault region is a design artifact intended to improve computational efficiency.

The module consists of two sections

- **Evaluate Darcy velocities and dispersion coefficients ('Interface with the Surrounding Geosphere' in the Theory Manual).**
- **Evaluate regionalized vault properties.**

```
⟨main⟩≡
        ⟨geosphere⟩
        ⟨regional⟩
        RETURN
        END
```

⋮

Evaluate components of Darcy velocity in rock for one sector (SEC).

The room axis is assumed to be parallel to the X component of the geosphere network cartesian coordinate system so the axial component is simply $V_R^A = V_x$.

The transverse groundwater velocity in the rock is correspondingly assumed to be in the YZ plane of the geosphere network cartesian coordinate system and is evaluated as $V_R^T = \sqrt{V_y^2 + V_z^2}$.

Define $\theta \equiv$ **angle between the axis of the room and the direction of water flow. Compute** $\sin(\theta)$ **and** $\cos(\theta)$.

Assume permeability of buffer is zero, and hence, Darcy velocity in buffer is zero $V_B^T = 0$.

```
⟨darcyComponents⟩≡
  C.....Compute axial and radial components of Darcy velocity
        DARKVA(SEC) = DARRKX(SEC)
        DARKVR(SEC) = SQRT(DARRKY(SEC)**2 + DARRKZ(SEC)**2)
  C.....Evaluate sin and cos of angle between room axis and flow
        RKVSIN = DARKVR(SEC) / DARRK(SEC)
        RKVCOS = DARKVA(SEC) / DARRK(SEC)
```

⋮

Figure 7: A portion of a TEXspec Subprogram Design Description

from the subprogram design description. Information for each subprogram on a structure chart is also extracted from the subprogram design description, ensuring that all design documentation is consistent.

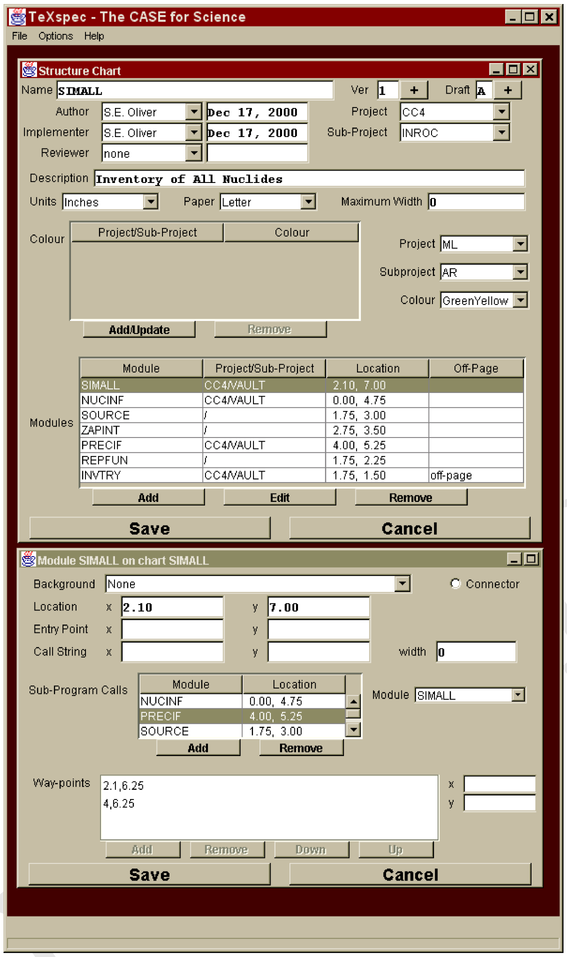
Data Dictionaries. TEXspec distinguishes between dictionary entries for requirements and design specification. Some design information is never applicable to requirements (e.g. a common storage block name).

It is possible to have a close corellation between entries in the requirements and design dictionaries. Design entries may optionally state a requirements dictionary entry which is related. When this is done, fields in the design dictionary aquire default values equivalent to the requirements data dictionary. This is particularly useful to inherit the mathematical symbol and description.

TEXspec produces a data dictionary listing which can show a cross reference of which products use which dictionary entries.

Graphical User Interface. There is a considerable amount of data contained in many plain text files in a typical TEXspec documented project. To assist users, a GUI has been developed as a Java application to act as a front end to the process. While there is little new technology embeded in the GUI, it is interesting to note that the GUI, at over 20,000 lines of code, is much larger that the TeXspec scripts.

An example screen is shown in figure 8. Here, the structure chart 'SIMALL' from figure 6 is being edited in the upper window. A subprogram on the chart is being modified in the lower window. The GUI can open many windows, so it is contained within an application desktop, which produces only one icon on the user's desktop.



Name SIMALL
Ver 1
Draft A
Author S.E. Oliver Dec 17, 2000
Implementer S.E. Oliver Dec 17, 2000
Reviewer none
Project CC4
Sub-Project INROC
Description Inventory of All Nuclides
Units Inches
Paper Letter
Maximum Width 0

Colour

Project/Sub-Project	Colour

Project ML
Subproject AR
Colour GreenYellow

Modules

Module	Project/Sub-Project	Location	Off-Page
SIMALL	CC4/VAULT	2.10, 7.00	
NUCINF	CC4/VAULT	0.00, 4.75	
SOURCE	/	1.75, 3.00	
ZAPINT	/	2.75, 3.50	
PRECIF	CC4/VAULT	4.00, 5.25	
REPFUN	/	1.75, 2.25	
INVTRY	CC4/VAULT	1.75, 1.50	off-page

Background None
Location x 2.10 y 7.00
Entry Point x y
Call String x y width 0

Sub-Program Calls

Module	Location
NUCINF	0.00, 4.75
PRECIF	4.00, 5.25
SOURCE	1.75, 3.00

Module SIMALL

Way-points
2.1,6.25
4,6.25
x
y

Figure 8: TeXspec GUI editing a Structure Chart

When editing a subprogram (module) on a structure chart, the user specifies the subprogram name, and sets a position in x,y coordinates. If the user wants to show an entry point on the chart to this subprogram (useful for charts with multiple entry points), then the location of the entry point must be specified. The location of the call string, and the maximum width of that string is also entered. Calls to other subprograms can be added from a dropdown list of all available modules. For each called subprogram, 'waypoints' determine the shape of the line connecting the two boxes.

Similar editing capability is provided for the overall structure chart in the upper window. Selecting a subprogram from the scroll list at the bottom of the upper window causes the lower window to appear.

Future Development

The next stage of TeXspec development will be to add some object oriented programming extensions, and a rudimentary interface between the GUI and an Integrated Development Environment (IDE) to assist in debugging and performance analysis.

The plain text files that store TEXspec data are formatted to be human readable and editable. This allowed TEXspec to be used before the GUI was developed. With the advent of a GUI to interface with this data, the file format may be redefined to an XML syntax.

The perl scripts may be reimplemented in Java, to permit a more seamless interface between the GUI and the main application.

The application may be divided into a client and a server. This would improve performance, assist in sharing data between users and projects, and provide more robust auditing and version tracking. The system could allow installation of files into a configuration management system. Dependencies between files would be monitored by the server, and ownership would be enforced.

A number of extensions may be made to the GUI, including preview capability for mathematical notations.

More diagram types and programming languages may be supported. In particular, object oriented diagrams may be added, and the full FORTRAN-9x, Java, or Perl syntax may be added.

The GUI support for the graphical products (Data Flow Diagrams and Structure Charts) could be based on editable graphics, or perhaps provide a 'preview' window. Having to process the file to see the format of the output is not a optimal.

Some allowance for formal tracing between design and requirements could be provided.

Conclusion

TEXspec provides a workable solution to computer aided software engneering requirements that are peculiar to scientific programs. It is a significant quality assurance device for these programs.

TEXspec is in use on several projects relating to modeling the disposal of Canada's nuclear waste. As such, it is a working tool, but is still in the early phases of development. Further enhancement will improve the capability of meeting quality assurance requirements imposed by standards such as CSA N286.7.

References

[1] J.J. Bunn. Floppy and flow user manual. 1997.

[2] B.W. Goodwin, T.H. Andres, D.C. Donahue, W.C. Hajas, S.B. Keeling, C.I. Kitson, D.M. LeNeveu, T.W. Melnyk, S.E. Oliver, J.G. Szekely, A.G. Wikjord, K. Witzke, and L. Wojciechowski. The disposal of canada's nuclear fuel waste: A study of postclosure safety of in-room emplacement of used candu fuel in copper containers in permeable plutonic rock. volume 5: Radiological assessment. Technical Report AECL-11494-5,COG-95-552-5, Atomic Energy of Canada Ltd, 1996.

[3] B.W. Goodwin, D.B. McConnell, T.H. Andres, W.C. Hajas, D.M. LeNeveu, T.W. Melnyk, G.R. Sherman, M.E. Stephens, J.G. Szekely, P.C. Bera, C.M. Cosgrove, K.D. Dougan, S.B. Keeling, C.I. Kitson, B.C. Kummen, S.E. Oliver, K. Witzke, L. Wojciechowski, and A.G. Wikjord. The disposal of canada's nuclear fuel waste: Postclosure assessment of a reference system. Technical Report AECL-10717,COG-93-7, Atomic Energy of Canada Ltd, 1994.

[4] D.E. Knuth. *Literate Programming*. Center for the Study of Language and Information, 1992.

[5] D.M. LeNeveu. Analysis specifications for the cc3 vault model. Technical Report AECL-10970,COG-94-100, Atomic Energy of Canada Ltd, 1994.

[6] Quality assurance of analytical, scientific, and design computer programs for nuclear power plants. Technical Report N286.7-99, Canadian Standards Association, 178 Rexdale Blvd. Etobicoke, Ontario, Canada M9W 1R3, 1999.

[7] S. Oliver, K. Dougan, K. Kersch, C. Kitson, G. Sherman, and L. Wojciechowski. Unit testing - a component of verification of scientific modelling software. In T.I. Oren and G.B. Birta, editors, *1995 Summer Computer Simulation Conference*, pages 978–983. The Society for Computer Simulation, 1995.

[8] N. Ramsey. Literate programming simplified. *IEEE Software*, September 1994.

[9] K. Rose. Very high level 2-dimensional graphics. In *1997 TeX User Group Conference*. TeX User Group, 1997.

[10] L. Wall, T. Christiansen, and R. Schwartz. *Programming Perl*. O'Reilly & Associates, 101 Morris Street, Sebastopol, CA 95472, second edition, 1989.

[11] E. Yourdon. *Modern Structured Analysis*. Yourdon Press/Prentice-Hall, 1989.