VARSKIN Software Modernization and Quality Assurance

RAMP User Group Meeting
Bethesda, MD
October 2019

Jeffrey M. Luitjens, Ph.D.
Senior Software Developer
jeff.luitjens@rcdsoftware.com

Work supported by the U.S. Nuclear Regulatory Commission (NRC), Office of Nuclear Regulatory Research (RES), under contract 31310018C0026
Introduction

- Software Quality Assurance Program
- VARSKIN Modernization Efforts
- Performance Tests
- Future VARSKIN Efforts and Vision
Software Quality Insurance Program
Software Quality Assurance Program

• A software quality assurance program was developed derived from the NRC Software Quality Assurance Program and Guidelines document\(^1\).

• The intent of developing such a program is to:
  • ensure a level of consistency between software projects and to
  • improve the quality of the developed software products ensuring they perform their defined functions correctly.

---

Software Life Cycle

• The software life cycle is defined by the following phases:
  • Requirements
  • Design
  • Implementation
  • Qualification Testing
  • Installation and Acceptance Testing
  • Use
  • Maintenance
  • Retirement
Testing

• Types of testing:
  • Qualification Testing – Formal testing that demonstrates the technical adequacy of the software product by ensuring it performs its function as defined by requirements.
  • Installation Testing – Installation of software on a system followed by the performance of acceptance testing to ensure proper functionality of a software product across many environments / platforms.

• Methods of testing:
  • Analysis without computer programs (hand calculations).
  • Comparison with other computer programs.
  • Comparison to experimental data.
  • Standard problems with accepted or known solutions.
  • Comparison to published data and / or correlations.
Configuration Control

• Established to ensure integrity and quality of a software project through the life cycle.
  • Identification and establishment of baselines.
  • Controlling and managing changes to the baseline.
  • Documenting changes to the baseline.
  • Verifying correctness of the new baseline prior to release.
Subversion Control

• The subversion repository is a remote repository that developers can link to.

• This allows the maintaining of a complete history of the software product with unique revision numbers.

• Allows developers to collaborate with the latest form of the code.

• Eases the process of source control.
Documentation and Deliverables

• Requirements document (or contracts) defining the requirements of the project.
• Theory manuals describing the design and basis of the code.
• Software release notes describing the implementation of methods by the developer.
• Software test report(s) identifying the test plan, methods, and results.
• User manuals describing use of the code.
• Source and executable code.
Error Reporting

• For software, reporting activities will be related to issues that may arise with use of codes by the user community.

• Users have the capability to identify and report errors to be dispositioned by a developer to determine if:
  • a change to the source code is required,
  • a change to a document is required to clarify use or intent of the software product, and/or,
  • a limitation needs to be identified with an acceptable workaround if one exists.
VARSKIN Modernization
Modernization Activities

• Conversion from Fortran fixed to free format.
• Three separate Fortran executables were combined into a single dynamically linked library (DLL) file.
• All variables are now explicitly declared and IMPLICIT NONE was invoked.
• Standard TYPE definitions were defined. This ensures consistent use of precision throughout the Fortran project. A precision of 15 digits is specified for REALs (double precision on most machines) with SDK (selected double kind) and INTEGERs were specified to contain 10 elements with SIK (selected integer kind).

! Specify appropriate precision to maintain 15
! decimal places of precision (double on most machines)
INTEGER, PARAMETER :: SDK = SELECTED_REAL_KIND(15)
INTEGER, PARAMETER :: SIK = SELECTED_INT_KIND(10)
Modernization Activities

• Constant PARAMETERS were defined throughout in a MODULE to ensure consistency and use of standard constants (e.g., PI, density of air).

• A general restructuring and functionalization of the Fortran source was performed. This is included functionalizing certain methods for organization and reuse.

• COMMON blocks and INTERFACE definitions are removed and shared routines are appropriately stored within modules and accessed through USE statements as needed.
Modernization Activities

• When whole numbers were applied as exponents they are now explicitly typed as integers (_sik). For example, instead of just 10.1**2 this was written as 10.1**2_sik to ensure the exact representation of the expression was evaluated versus the approximation when the compiler treats 2 as a REAL. Using ‘_sik’ here increases the accuracy and speed of the calculation.
Modernization Activities

• Archaic calls to the DOUBLE TYPE intrinsic functions were removed. For example, if DCOS(x) was called, this was changed to COS(x) in which compilers now provide an output of the same type as x to ensure consistency.

• Coding has been updated to use REAL(X, sdk) where X is an integer to be converted to a REAL of double precision specified by KIND=sdk. This differs from FLOAT(X) which converted an integer to a REAL(4) which is single precision.
Modernization Activities

• Compiler options for the Fortran project were set based on best practices for compilation with the Intel Fortran compiler version 19.0.
  • The compiler option for checking array bounds was activated. This check occurs at runtime and thus an error will be thrown only while executing the program. This check cannot be performed on compilation and thus cannot be guaranteed to catch all potential array bounds issues, only extensive use of the program can do this.
Modernization Activities

• Previously, coding was written and developed in Fortran 77 making it out of date with current accepted standard.

• As of now, the Fortran source is now compliant with the 2015/2018 Fortran Standard as approved by the International Organization for Standardization (ISO). This involved moving away from use of archaic functions (e.g., DFLOAT), ensuring that actual arguments being passed were consistent with dummy arguments, and modifying how variables are declared, etc.

• This works towards maintainability and consistency of the code improving its overall quality.
Performance Tests
Performance Tests

- A performance test utilizing volume averaging was performed to determine if improvements to the run time of the code have improved.
- A 2.5 increase in runtime efficiency.
- Adding nuclide now takes 40% less execution time.

<table>
<thead>
<tr>
<th>Version</th>
<th>Run Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>v5.3</td>
<td>125</td>
</tr>
<tr>
<td>v6.0</td>
<td>120</td>
</tr>
<tr>
<td>v6.1</td>
<td>120</td>
</tr>
<tr>
<td>v6.2</td>
<td>120</td>
</tr>
<tr>
<td>v6.2.1</td>
<td>50</td>
</tr>
</tbody>
</table>
VARKSIN Future Efforts and Vision
VARSKIN 7.0

• User interface will receive an upgraded more modern face lift.
  • Computational calculations will remain in Fortran.
  • User interface will be ported to Java.

• Consideration being given to:
  • Live updating of results as inputs change.
  • Allowance of parallel computations.
  • Built-in integrated testing that can be executed automatically with formal report generation for user to confirm proper installation.
  • Results export, CSV, formal reports, and etc.

• Addition of new models:
  • Neutron dosimetry
  • Alpha dosimetry
  • Eye dose model