LLNL-SM-511465

Software Development Plan

Gregory Pope

V 1.2

May 9, 2012

Disclaimers

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Approvals

Author:

_____ _____

Date

Gregory Pope
CCSI SQA

Approve:

_____ _____

Date

Paolo Calafiura
Team 8 Lead

Concur:

_____ _____

Date

David Miller
CCSI Program Director

Revision History

Document Version	Revision Date	Originator(s)	Revision Description
V1.0	10/19/2011	G. Pope	First edition
V1.1	12/19/2011	G. Pope	Minor updates

Table of Contents

1. Introduction

This Software Development Plan (SDP) describes how the Carbon Capture Simulation Initiative (CCSI) software and modeling activities will meet or exceed future software quality assurance goals that inevitably will be established if the initial phase modeling and simulation efforts are successful. Specifically this document will describe, for each of the ten DOE Work Activities described in DOE Order 414.1-d, the processes (consisting of activities, tools, work products and artifacts) which are implemented by CCSI to develop software and how they meet the required level of rigor based on risk level using a graded approach. To assure the integrity and repeatability of early work done with statistical or mathematical based research tools such as R, MFIX, ASPEN, and Matlab, a good scientific peer review process, such as ANSI/ASQ Z.13-1999, will be used. This Software Development Plan is currently not required by the DOE FE office, but is being prepared pro-actively so that software and models developed in the early phases of the CCSI project can comply with anticipated DOE standards and best practices for research development. This SDP will also hopefully be useful for determining the pedigree of the CCSI codes and models and the known risks inherent in using them for decision informing.

1.1 CCSI Project Overview

The *Carbon Capture Simulation Initiative* (CCSI) is a partnership among national laboratories, industry and academic institutions that will develop and deploy state-of-the-art computational modeling and simulation tools to accelerate the commercialization of carbon capture technologies from discovery to development, demonstration, and ultimately the widespread deployment to hundreds of power plants. By developing the *CCSI Toolset*, a comprehensive, integrated suite of validated science-based computational models, this initiative will provide simulation tools that will increase confidence in designs, thereby reducing the risk associated with incorporating multiple innovative technologies into new carbon capture solutions. The scientific underpinnings encoded into the suite of models will also ensure that learning will be maximized from successive technology generations.

The CCSI Task Teams are responsible for the design, implementation, verification and validation of simulation of software codes that support Carbon Capture Simulations which allow new concepts to move from the lab to power plant quickly and at low cost. The DOE FE SQA requirements have yet to be specified for the CCSI program. As a result the CCSI program has proactively organized around best practices for software development as described in this Software Development Plan (SDP) and on the CCSI Best Practices website https://www.acceleratecarboncapture.org/drupal/wiki/ts8/best-practices. There are ten task teams currently in the CCSI project as shown in figure 1:

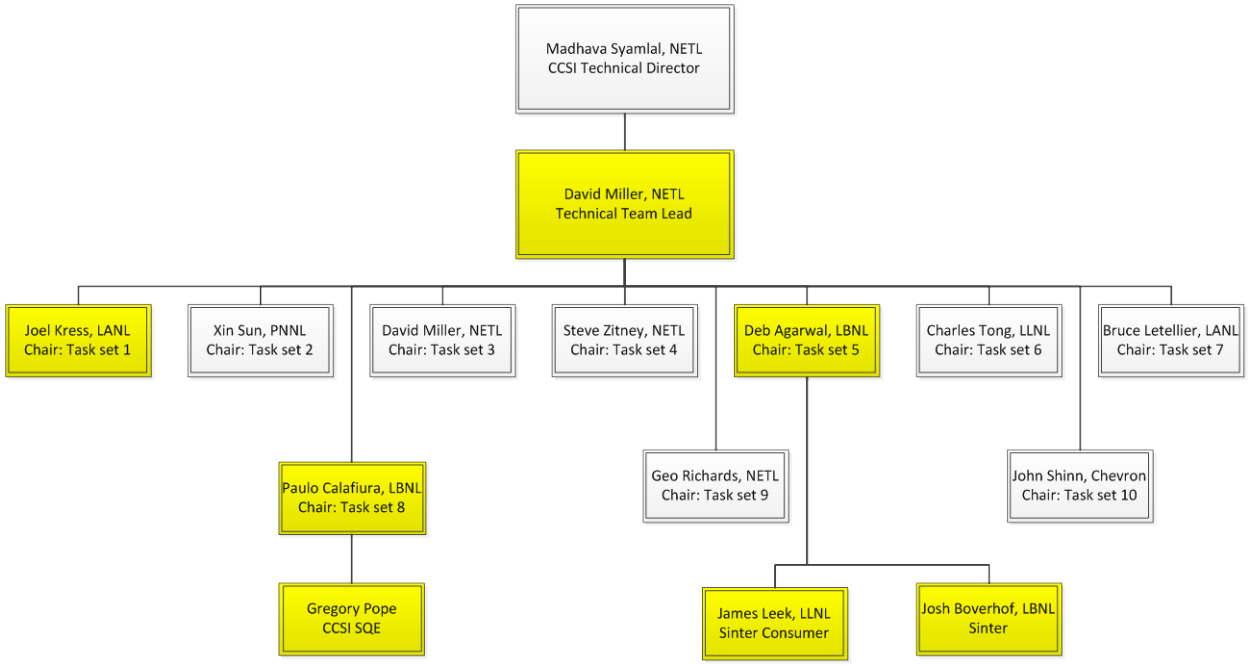


Figure 1. CCSI Organizational Chart

1.2 Purpose and Scope of the CCSI SDP

The purpose of this software development plan is document the pedigree of the early stage research software and models being developed and used as if DOE Oder 414.1-d was applied. DOE Order 414.1-d includes a graded approach which will be used to risk grade codes and strike a balance between agility and discipline required for effective scientific discovery. The early stages of the CCSI research and development codes will allow maximum agility with light weight discipline and workflow implemented via automated tools which improve developer productivity. As the research codes mature and are used for decision informing, licensing, and safety related applications the risk grade of the codes and appropriate rigor will be applied. Software quality assurance oversight for CCSI is the responsibility of the CCSI Software Quality Engineer residing in Task Team eight. The CCSI Software Quality Engineer is responsible for developing the CCSI SDP, risk grading the CCSI codes with assistance from the CCSI developers, and generally assisting the CCSI collaborating National Laboratories and partners with appropriate Software Quality Assurance practices and tools.

In addition to new code development in task team 5, statistical modeling is occurring in Task team 1, 2, 3, 4, 6, 7, and 8. The work being done using a number of open source, National Laboratory, and commercial tools. The iterative modeling taking place with these tools does require a good peer review process of the data inputs, and when important results are achieved, appropriate version information and data is archived to be able to repeat the results.

1.3 CCSI Software Development and Modeling Activities

Most of the task teams (1 through 8) have either software development activity (task team 5) or modeling (Teams 1, 2, 3, 4, 6, 7, 8) occurring in them. The modeling is performed using open source or commercial modeling tools or spreadsheets. The software for these modeling tools has already been developed by open source or commercial vendors and is not covered by this SDP. The data or coding of the models used as inputs to these tools is however covered by this SDP. Task team five is writing software to interface various tools together; this new code is covered by the SDP.

1.3.1 **Task Team One**, Basic Data and Models, is developing models that can be embedded into models used by tasks 2-4. The team is using the open source statistical modeling tool R version 2.1.3.2 to develop models. Models are developed using a C like language and provided libraries.

1.3.2 **Task Team Two**, Particle and Device Scale Models, is using MFIX (a Fortran Multiphase Flow with Interphase eXchange) and Ansys Fluent , which are Computational Fluid Dynamics (CFD) codes.

MFIX is a general purpose computer code developed at the National Energy Laboratory (NETL) for describing the hydrodynamics, heat transfer, and chemical reactions in fluid systems. MFIX has been used for describing bubbling and circulating fluidized beds and spouted beds. MFIX calculations give transient data on the three dimensional distribution of pressure, velocity, temperature, and specialized mass fractions. MFIX code is based on a generally accepted set of multiphase flow equations. The code is used as a "test-stand" for testing and developing multiphase flow constructive equations. The tool requires the user to supply data values for the various required input parameters.

ANSYS FLUENT software contains the broad physical modeling capabilities needed to model flow, turbulence, heat transfer, and reactions for industrial applications ranging from air flow over an aircraft wing to combustion in a furnace, from bubble columns to oil platforms, from blood flow to semiconductor manufacturing, and from clean room design to wastewater treatment plants. Special models that give the software the ability to model in-cylinder combustion, aeroacoustics, turbomachinery, and multiphase systems have served to broaden its reach. The tool requires the user to supply data values for the various required input parameters.

1.3.3 **Task Team Three**, Process Synthesis and Design, is primarily using the Aspen Custom Modeler (ACM) and GAMS (General Algebraic Modeling System).

Aspen Custom Modeler (ACM) is used to quickly create rigorous models of processing equipment and to apply these equipment models to simulate and optimize continuous, batch, and semi-batch processes. It is used across many industries, including chemicals, power, nuclear, food and beverage, metals and minerals, pharmaceuticals, and

consumer goods. Aspen Customer Modeler is a core element of AspenTech's aspenONE® Engineering applications.

The General Algebraic Modeling System (GAMS) is a high-level modeling system for mathematical optimization. GAMS is designed for modeling and solving linear, nonlinear, and mixed-integer optimization problems. The system is tailored for complex, large-scale modeling applications and allows the user to build large maintainable models that can be adapted to new situations. The system is available for use on various computer platforms. Models are portable from one platform to another.

The GAMS algebraic modeling language (AML) is formally similar to commonly used forth generation programming languages. GAMS contains an integrated development environment (IDE) and is connected to a group of third-party optimization solvers. Among these solvers are BARON, COIN solvers, CONOPT, CPLEX, DICOPT, GUROBI, MOSEK, SNOPT, and XPRESS.

GAMS facilitates the users to implement a sort of hybrid algorithms combining different solvers in a seamless way. Models are described in concise algebraic statements which are easy to read, both for humans and machines. NETL has also developed some interface code among these tools, although the majority of this work is in Task 5.

1.3.4 Task Team Four, Plant Operations and Control, is developing dynamic models in ACM and Aspen Dynamics.
Aspen Plus Dynamics extends Aspen Plus steady-state models into dynamic process models, enabling design and verification of process control schemes, safety studies, relief valve sizing, failure analysis, and development of startup, shutdown, rate-change, and grade transition policies. Aspen Plus Dynamics is a core element of AspenTech's aspenONE® Engineering applications.

1.3.5 Task Team Five, Integration Framework, currently developing interfaces among tools and tools for building reduced order models. See figure 2 below. Explanation is in section 1.4.

1.3.6 Task Team Six, Uncertainty Quantification and Optimization is focusing on UQ, and using PSUADE a C++ based UQ tool developed by Charles Tong of LLNL. The PSUADE code has been subjected to static analysis using the Klocwork static analyzer tool. The findings are being evaluated. The code is kept in a subversion repository and is tested with a partially completed set of verification tests. The code is used on the ASCEM, NRAP. and ASC program for UQ,

1.3.7 Task Team Seven, Risk Analysis and Decision Making, is developing decision tools. Currently, these are largely spreadsheet models with some underlying Matlab.

MATLAB (matrix laboratory) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.

1.3.8 Task Team Eight, Software Development Support, is creating a Drupal wiki to support collaboration, a Subversion code repository, and Trac and Jira trackers for defect tracking and user story tracking. The SVN repository is being used for the Drupal web site software Configuration Management, the Jira trackers are being used by Team 8 to add epics, user stories, and tasks to the team's backlog.

> Drupal is a free and open-source content management system (CMS) and content management framework (CMF) written in PHP and distributed under the GNU General Public License.[2][3][4] It is used as a back-end system for at least 1.5% of all websites worldwide[5][6] ranging from personal blogs to corporate, political, and government sites including whitehouse.gov and data.gov.uk.[7] It is also used for knowledge management and business collaboration.
>
> Apache Subversion (often abbreviated SVN, after the command name *svn*) is a software versioning and a revision control system distributed under a free license. Developers use Subversion to maintain current and historical versions of files such as source code, web pages, and documentation. Its goal is to be a mostly-compatible successor to the widely used Concurrent Versions System (CVS).
>
> JIRA is a proprietary issue tracking product, developed by Atlassian, commonly used for bug tracking, issue tracking, and project management. The product name, JIRA, is not an acronym but rather a truncation of "Gojira", the Japanese name for Godzilla.[3] It has been developed since 2004.

1.3.9 Task Team Nine, Industrial Challenge Problems, is not currently developing software or using modeling tools.

1.3.10 Task Team Ten, Industrial Collaboration, is not currently developing software or using modeling tools.

1.4 Organization of CCSI Task Set Five Codes

The following block diagram (figure 2) portrays the CCSI research codes currently under development in task 5.

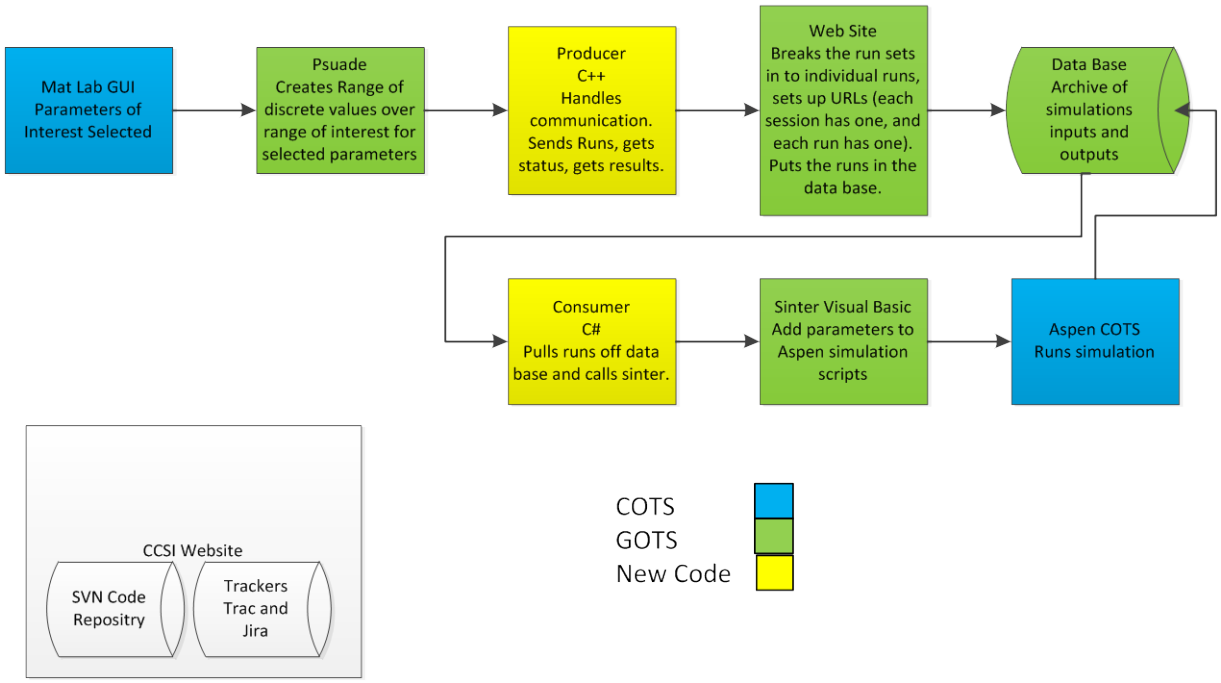


Figure 2.

The primary purpose of the CCSI research codes under development are to provide interfaces or “glue” for existing COTS (Commercial off the shelf) and GOTS (Government of the Shelf) codes to operate with each other. The CCSI created tool chain allows parameters of interest to be selected using the GUI interface of MatLab (a COTS tool). MatLab provides the user selected input parameters to Psuade, an existing uncertainty quantification tool developed at LLNL. Psuade creates a range of values for the selected parameters of interest. The input value ranges are sent from Psuade to the Communicator tool. Communicator is a new C++ code that can communicate with the existing CCSI website to set up simulation runs, monitor run status, and get results. The CCSI web site will create individual runs, breaking the run sets into individual runs. The individual runs are put into a data base. The individual runs in the data base are collected by a new software code called the consumer. The consumer code sets up spreadsheet like formats for input into Sinter, which is an existing tool allowing spreadsheets to be used as input into Aspen. Using the Sinter supplied input parameters; the Aspen tool (existing COTS tool) will do the simulation and place the results on the data base, where they can be queried by the Communicator tool. The purpose of this tool chain (collectively called Sinter) is to facilitate an automated way to determine the influence of selected parameters on the

simulation. This information will help determine and quantify the uncertainty of the results obtained from the Aspen based simulations.

2 Project Risk and Grading

The Task set 5 primary developer and the CCSI Software Quality Engineer have determined the CCSI risk classifications or risk levels (RL). The CCSI software codes have been previously assessed in accordance with the CCSI SQAP and documented using the Risk Grading Tool[1] to have a risk level of 4, 1 being the highest (safety code) and 5 being the lowest (no severity or consequence of failure). Risk level 5 codes have the lowest severity consequence of failure; risk level 3 codes have a moderate consequence of failure; risk level 2 codes are safety significant severity or consequence of failure. The risk grading tool results are shown as screen shots in appendix B. These results are archived in the Risk Grading Tool[2].

1. Risk Grading Tool currently located at LLNL internal website: https://caribou-r.llnl.gov/projectManager/?proj_type=grading. Tool to be available through CCSI web site in the future.
2. Ibid

Practice Rigor Levels

Rigor Level	Definition
Managed 'M'	Highest level of rigor or formality. The project has processes and work products, which are reviewable in advance and well documented. Artifacts are living documents that are integral with the life cycle and updated as needed. Work products and processes adhere to templates and standards used. Appropriate level of planning is done and progress is measured against the plan. Appropriate measurements of the processes and products are made to support management decisions.
Documented 'D'	Middle level of rigor or formality. Artifacts are purposefully produced to communicate decisions during the course of the project. Activities may be combined and documented together. When appropriate, plans may be produced prior to activities.
Understood 'U'	Lowest level of rigor or formality. These practices may be implemented either formally or informally. Any produced artifacts are characterized as incidental, such as e-mail discussions and the like. Informal in nature.

Practice Implementation Levels

Implementation Level	Definition
Full 'F'	Full implementation of the indicated rigor level is required. Many practices have multiple options for implementation at any given rigor level, the specific implementation is left to the project team to decide. Greater rigor may be applied. If there is a strong case for not implementing a practice, the justification should be documented.
Tailored 'T'	Tailored implementation of a practice allows for variation in the indicated rigor level. Tailoring is also adapting the practice to the level of development control. The rigor levels above and below the indicated rigor level should bound the tailored implementation. A practice may be eliminated if there is a strong case for doing so.

3 Required Software Quality Practices Mapped To Waste IPSC

The following tables lay out the software Work Activities and the documentation associated with the required activities. The Rigor indicators are as follows:

FM – Fully implemented and Managed

TM – Tailored and Managed

FD – Fully implemented and Documented

TD – Tailored and Documented

FU – Fully Understood

TU – Tailored and Understood

N/A – Does not apply (RL 5 only)

SQA Work Activity 1 - Software Project management and Quality Planning

Activity	Sinter Level 4	CCSI Implementation
Plan and manage project activities, resources and commitments (including schedule; budget; software development methodology; customer interactions; feedback and status reporting; and identify, acquire, and deploy resources such as development and test environments)	**TU**	Internal Task team SCRUM meetings, stakeholder requirements entered as user stories added to Jira tracker. PI meetings twice yearly to report progress. Collaborative web site which includes tasks and assignments.
Determine applicable regulatory requirements, indentify organization, identify Quality Assurance Program and Records, and identify auditing process.	**TU**	CCSI Software Development Plan defines risk and rigor level used for software. Risk Grading tool defines Risk Level . CCSI Software Development Plan (SDP) defines how rigor levels met

Activity	Sinter Level 4	CCSI Implementation
Select and utilize standards	**TD**	CCSI using DOE Order 414.1-d
Identify, define, and plan software quality assurance activities	**TU**	Contained in this Software Development Plan (SDP)
Implement process improvement activities	**TU**	Submitted as User Stories to TRAK tracker, implemented by task teams.

SQA Work Activity 2 – Software Risk Management

Plan and execute a risk management process (including risk analysis and mitigation)	**TU**	Stakeholders may enter risks as user stories onto TRAK tracker. Risks and mitigations also addressed in weekly SCRUM teleconferences.

SQA Work Activity 3 – Software Configuration Management

Plan and implement a software configuration management process (including disaster recovery planning and release version control, document control, purchased items, instructions, procedures, drawings)	**TU**	SVN tool used. Open source distributed Software CM tool.

SQA Work Activity 4 - Procurement and Supplier Management

Activity	Sinter Level 4	CCSI Implementation
Implement processes for controlling interactions with subcontract interfaces	**TD**	Using DOE approved Procurement Plan
Implement processes for controlling interactions with collaborations and tool vendors	**TD**	CCSI wiki allows collaboration between labs. Best Practices wiki links to tool vendors.
Include software quality requirements in procurement and selection process	**TD**	Acquisition Plan contained in DOE approved Procurement Plan
Qualify software for intended usage (may be included in V&V)	**TU**	See V&V

SQA Work Activity 5 - Software Requirements Identification and

Indentify (to the extent known) review and update requirements and maintain traceability of requirements	**TU**	Requirements, features, elicited via user stories on TRAK tracker. Also elicited at PI meetings.
Identify and track technical constraints	**TU**	Language is C++ and C# for CCSI generated “glue” code. Using ASPEN Plus simulation tool (COTS tool) , Psuede (LLNL UQ code), MatLab (COTS tool)

SQA Work Activity 6 - Software Design and Implementation

Activity	Sinter Level 4	CCSI Implementation
Identify the software design (to the extent known, including software and system interfaces)	**TU**	Requirements go directly to code. Code is the design.
Identify and utilize coding standards	**TU**	C++ and C# standards used as appropriate. Code also is analyzed by Klocwork static analyzer which is capable of finding structural defects the compilers and testing may miss..

SQA Work Activity 7 - Software Safety

Software Safety Plan (including safety and hazard analysis)	**TU**	Currently all codes are used at R&D level only.
Perform reliability, vulnerability and usability analysis as needed	**TU**	Currently all codes are used at R&D level only. Software Safety Plan to be written later if required.
Designation for intended usage and users	**TU**	Currently all codes are used at R&D level only. Codes are intended for use by expert users who can evaluate the significance of results.

SQA Work Activity 8 - Verification and Validation

Activity	Sinter Level 4	CCSI Implementation
Conduct software testing (unit, integration, system)	**TU**	Tests developed for demonstration of the prototype to stakeholders.
Conduct walkthroughs / desk checks and peer reviews.	**TU**	Ad hoc for software Use ANSI/ASQ Z1.13-1999 as guidance for research reviews of modeling data.
Implement software integration and build processes	**TU**	Appropriate build tools used. Using Eclipse IDE and compatible plug ins.

SQA Work Activity 9 - Problem Reporting and Corrective Action

Implement problem reporting and resolution (corrective action) processes	**TU**	Problem reporting done via TRAC, soon to be Jira (open source bug tracing tool)
Provide maintenance support as needed	**TU**	Developers support code maintenance and users as required.
Create release notes as needed	**TU**	Release notes in email to users and in code comments.

SQA Work Activity 10 - Training of Personnel in the Design, Development, Use, and Evaluation of Safety Software

Activity	Sinter Level 4	CCSI Implementation
Select, train, and task team members (including training in software safety design, development, and evaluation)	**TD**	Team includes highly skilled domain experts and computer scientists.
Produce user and installation guidance as needed	**TU**	To be completed as needed. Communicated via SCRUM meetings and emails.

4 Method of Meeting Activity

4.1 Software project management and quality planning (complies):

The two person Sinter code team, members of Test set five, are located at LBNL and LLNL and communicate as needed via email. They virtually attend Task set five SCRUMS weekly. Quality planning will use this CCSI SDP and a risk grading tool. The risk grading tool has assessed the CCSI effort in Task set five as risk level 4. Tasks for the software development are allocated via trackers. Currently Trac is used; however Jira is also under evaluation. While Jira may be a better defect tracking tool, it is more complex for management applications. So task trackers that are part of Drupal such as Case Tracker Plus are currently being evaluated as well.

4.2 Software risk management (complies):

Risks to the CCSI program can be entered into the tracker and reviewed on a regular basis. One of the identified risks is developing software without a governing DOE standard. To mitigate this risk this Software Development Plan is being written. A second risk will be if the research codes are successful and continue on. If this happens a number of risk mitigations may have to be used, such as beefing up the test coverage in the domains of interest, more input checking code added, additional error messages and exception handling added, porting to other platforms, peer review of critical algorithms and parameters, and static analysis of the code. Another risk is to over burden the research phase of the program with heavy weight and restrictive practices. To mitigate this risk an Agile methodology is being used with a graded approach.

4.3 Software configuration management (complies):

The CCSI codes will be kept in an open source SVN repository, https://www.acceleratecarboncapture.org/drupal/all-ccsi/subversion . The SVN repository will assure version control on the software, and that multiple team members can modify and test code without interference to/from other developers. The repository is automatically backed up periodically. Periodic archiving of the R tool data or Matlab data should be accomplished to preserve version numbers and data integrity. This does not have to be done continuously, but certainly after the important modeling milestones are achieved, so that the results can be repeated.

4.4 Procurement and supplier management (complies):

It is envisioned that some COTS software such as Aspen and Matlab will be procured by CCSI. The procuring national laboratory will use a DOE approved procurement system to support the CCSI effort.

4.5 Software requirements identification and Management (complies):

Requirements for the CCSI software are elicited s use cases via trackers on the CCSI collaborative web site, https://svn.acceleratecarboncapture.org/trac . It therefore sets the baseline expectations and to some extent determines the features required in the software being developed. The stakeholders may also make their requirements known to the CCSI developers through PI meetings and other forms of communications.

4.6 Software design and implementation (complies):

The prototype will proceed from requirements directly to code. Code architecture is designed to interface existing tools to be used in a tool chain. The CCSI developed code will be "glue" code to enable existing components to communicate more efficiently. The design of the user interface is dictated by the existing components.

4.7 Software safety (complies):

The research codes are not currently being used in a facility safety application or to inform decisions on facility design. As the CCSI code consequences of failure increase the required levels of rigor for development will also increase.

4.8 Verification and validation (complies):

The CCSI software will develop ranges of inputs for selected key parameters. These ranges will create individual simulation runs on the Aspen simulator. After verification of Aspen results by domain experts or validation of results against experiments these simulation runs will be part of the verification suite of tests that are run nightly or after changes are detected in the source codes. The test cases will also be run periodically as regression tests at the system level. Unit tests are envisioned to be created along with the code using a framework such as cppunit, these tests may also be used as part of the regression test suite.

4.9 Problem reporting and corrective action (complies):

Problem reporting will be accomplished using the CCSI defect tracker, currently Trac, but Jira is under consideration to replace it. The tool can be found at , https://svn.acceleratecarboncapture.org/trac . The modelers and development team are distributed among the participating National Laboratories with commercial and academic partners. The team collaborates using a Drupal wiki hosted at LBNL, https://www.acceleratecarboncapture.org/drupal/ .

4.10 Training of personnel in the design, development, use, and evaluation of safety software (complies):

The development staff for the CCSI has in depth academic backgrounds and research and development experience in the knowledge domains required for development of

their respective codes and models (physics, chemistry, computer science, CFD codes, etc.). Code users can self train using supplied primers, demo problems, tutorials and sample problems. Personnel files documenting the academic credentials and R&D experience of team members meet are available.

4.11 Important Data

The other major software related activity is modeling. Modeling differs from software development as the software based tools used for doing the modeling have already been developed. What the CCSI task teams engaged in modeling are doing is providing inputs to these modeling tools. Inputs can be data values and/or programming language statements. CCSI has little to no control over the code development of these existing tools; however CCSI does have control over how the inputs used with the modeling tools are handled. As part of good scientific process, important results obtained from the modeling tools should preserve the data used and the version of the tools used so that the modeling results can be repeated in the future. Also, important modeling discoveries should require peer review of results to assure that the result was obtained for the right reason. It is understandable that a single researcher using a modeling tool may make many trial and error changes during the model development. Each of these variations do not need to be peer reviewed or archived, However, when the researcher does develop a model that is significant and may inform other models or algorithms to be added to the code, then a peer review and archiving of results would seem appropriate. Archiving and peer review of data used as inputs to modeling tools can be accomplished following the practices stated in ANSI/ASQ Z1.13, *Quality Guidelines for Research.* Good practices for modeling and for spreadsheets are listed in Appendix c and D respectively

Appendix A Acronyms, Abbreviations, and Terms

Activity	Executing individual actions needed to fulfill a practice. For instance, coding activity fulfills the practice of developing code modules as does compiling activity and debugging activity.
Artifact	A report, log, e-mail, or other record that is created as a result of a software development activity. For example, filling out a defect report on a defect-tracking tool such as Bugzilla or ClearQuest creates an artifact defect report. Running a set of test cases may create a test log artifact. Compiling code creates an artifact listing. Writing down notes from a design discussion creates a design artifact.
Consequence Tier	See **Error! Reference source not found.**, **Error! Reference source not found.**.
COTS	Commercial off-the-shelf
DOE	Department of Energy
DSA	Documented Safety Analysis
Graded Approach	The process of ensuring that the level of analysis, documentation, and actions used to comply with a requirement in this part is commensurate with: (1) The relative importance to safety, safeguards, and security; (2) The magnitude of any hazard involved; (3) The life cycle stage of a facility; (4) The programmatic mission of a facility; (5) The particular characteristics of a facility; (6) The relative importance of radiological and non-radiological hazards; and (7) Any other relevant factor.
Grading	The process by which the level of detail in analyses, documentation, and actions necessary to comply with requirements is commensurate with: • The relative importance to safety, safeguards, and security; • The magnitude of any hazard involved; • The life cycle stage of a facility; • The programmatic mission of a facility; • The particular characteristics of a facility; and • Any other relevant factors.
Hazard Analysis	The determination of material, system, process, and plant characteristics that can produce undesirable consequences, followed by the assessment of hazardous situations associated with a process or activity. Largely qualitative

	techniques are used to pinpoint weaknesses in design or operation of the facility that could lead to accidents. (D&P Manual of NWC)
IEEE	Institute of Electrical and Electronics Engineers, Inc.
Inspection	1. Measuring, examining, testing and gauging one or more characteristics of a product or service and comparing the results with specified requirements to determine whether conformity is achieved for each characteristic. [ASQ] 2. A static analysis technique that relies on visual examination of development products to detect errors, violations of development standards and other problems. Types include code inspections; design inspection. [IEEE 610] An inspection is a formal method of review where a team of peers, including the author, meets to examine a work product. An inspection usually consists of a kick-off meeting followed by the inspection review. The work product is typically inspected when the author thinks it is complete and ready for transition to the next phase or activity. The inspection may be a project milestone. The focus of an inspection is only on defect identification. Individual preparation using checklists and assigned roles is emphasized. Metrics are collected and used to determine entry criteria in the inspection meeting as well as for input into product/process improvement efforts. [Westfall] Inspection roles include: Moderator, Scribe, Author, and Inspector.
NSQAP	CCSI Software Quality Assurance Plan
Performance	1. The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage. 2. A measure relating to time. The actual definition of *performance* is project-specific. For some projects, this may mean speed of processing to arrive at an answer or create a display. For others, it may mean transaction throughput. For still others, it may mean response time under various levels of system loading.
Practice	Actual performance or application of a group of activities. For instance, the practice of analyzing requirements may involve many activities such as enumeration, abstraction, prioritization, assignment, verification, etc.
Practice level “Documented”	The middle level of rigor or formality. Artifacts are produced to communicate decisions during the course of

	the project. The project follows the goals of the practice with limited documentation of work products and activities. Activities may be combined and documented together. When appropriate, plans may be produced prior to activities.
Practice level "Managed"	The highest level of rigor or formality. The project has processes and work products, which are reviewable in advance and well documented. If templates and standards are used, the work products and processes adhere to them. Appropriate level of planning is done and progress is measured against the plan. Requirements are traceable to code and test cases. Appropriate measurements of the processes and products are made to support management decisions.
Practice level "Understood"	The lowest level of rigor or formality. The project is not required to implement these practices but may choose to do so, either formally (managed) or informally (documented). Any produced artifacts are characterized as incidental, such as e-mail discussions and the like. Ad hoc in nature.
Practices	Requirements employed to prescribe a disciplined uniform approach to the software development process
Process	A set of interrelated work activities characterized by a set of specific inputs and value added tasks that make up a procedure for a set of specific outputs.
Professional Judgment	The process of forming an opinion or evaluation by discerning and comparing that the software process and/or software product results conform to the technical or ethical standards of the applicable professions within the context of the environment in which it will be used (e.g., software engineering, computer science, physics, chemistry, seismology, meteorology, PMBOK®, SWEBOK®, etc.).
QA	Quality Assurance
Quality Assurance	All those actions that provide confidence that quality is achieved.
Regression Testing	Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.
Risk	The quantitative or qualitative expression of possible loss that considers both the probability that an event will occur and the consequence of that event. [DOE 5480.23]

Safety and Hazard Analysis Software and Design Software	Software that is used to classify, design, or analyze nuclear facilities. This software is not part of an SSC but helps to ensure the proper accident or hazards analysis of nuclear facilities or an SSC that performs a safety function. (See DOE O 414.1C)
Safety Management Program	A program designed to ensure a facility is operated in a manner that adequately protects workers, the public, and the environment by covering topics such as quality assurance; maintenance of safety systems; personnel training; conduct of operations; inadvertent criticality protection; emergency preparedness; fire protection; waste management; or radiological protection of workers, the public, and the environment. (10 CFR 830)
Safety Software	Includes the following: • Safety System Software • Safety and Hazard Analysis Software and Design Software • Safety Management and Administrative Controls Software
Safety System Software	Software for a nuclear facility (including radiological) that performs a safety function as part of an SSC and is cited in either 1) DOE approved Documented Safety Analysis or, 2) an approved hazard analysis. (See DOE O 414.1C)
Scalability	The ability of a system to handle increased capacities of users, devices, interfaces, storage, simultaneous processes, or threads without degrading below a desired performance level.
Security	The ability of the system to protect classified data from compromise by unauthorized sources, the ability of the system to protect data from inadvertent loss or modification, the ability of the system to protect against deliberate attempts to inject faults, or attempts to gain unauthorized control of the system.
Software	Computer programs, procedures, and associated documentation and data pertaining to the operations of a computer system. (See DOE O 414.1C and NQA-1-2000)
Software Life Cycle	1. All activities (and work products) required to analyze, define, develop, test, and deliver a software product. A high-level representation of the phases of the software development process. 2. The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase. *Note:* These phases

	may overlap or be performed iteratively. [IEEE 610]
Software Quality Assurance	A planned and systematic approach to the evaluation of the quality of and adherence to software product standards, processes and procedures. Software Quality Assurance includes the process of assuring that standards and procedures are established and are followed throughout the software acquisition life cycle.
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
SSC	Structure, system, or component
STPA	System Theoretic Process Analysis – A newer hazard analysis technique developed by Dr. Nancy Leveson especially designed for systems controlled by software. Handles multiple simultaneous failures and user errors.
Technical Safety Requirements	The limits, controls, and related actions that establish the specific parameters and requisite actions for the safe operation of a nuclear facility and include, as appropriate for the work and the hazards identified in the documented safety analysis for the facility: safety limits, operating limits, surveillance requirements, administrative and management controls, use and application provisions, and design features, as well as a bases appendix. (10 CFR 830)
Traceability	The degree to which a relationship can be established between two or more products of the development process; especially products having a predecessor, successor, or master subordinate relationship to one another.
Usability	The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.
Verification and Validation	The process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements. [IEEE 610]
Walkthrough	A static analysis technique in which a designer or programmer leads member of the development team and other interested parties through a segment of documentation or code, and the participants ask questions and make comments about possible errors, violation of development standards and other problems. [IEEE 610] A walkthrough is a method of peer review. A team of peers meets with the author to examine a work product and provide feedback. This is typically done on the first draft of a document or after a clean compile of the code. The focus is on a general evaluation and may include: a review of

	style and an analysis of engineering choices, as well as defect identification. Walkthrough roles include: Moderator, Scribe, Author, and Reviewer(s). Typically, the author presents (walks through) the work product and explains it to the team. Preparation before the meeting is less emphasized than in inspections. [Westfall] The purpose of a systematic walkthrough is to evaluate a software product. A walkthrough may be held for the purpose of educating an audience regarding a software product. The major objectives are to find anomalies, improve the software product, consider alternative implementations, and evaluate conformance to standards and specifications.
Work Product	A document or any tangible item created before, during, or after engaging in a software development activity. The work product may or may not be automatically created by the activity. For instance, a listing of test cases, a Software Quality Assurance Plan, Action Item list from a project meeting, or a list of desired features. Artifacts are also considered work products.

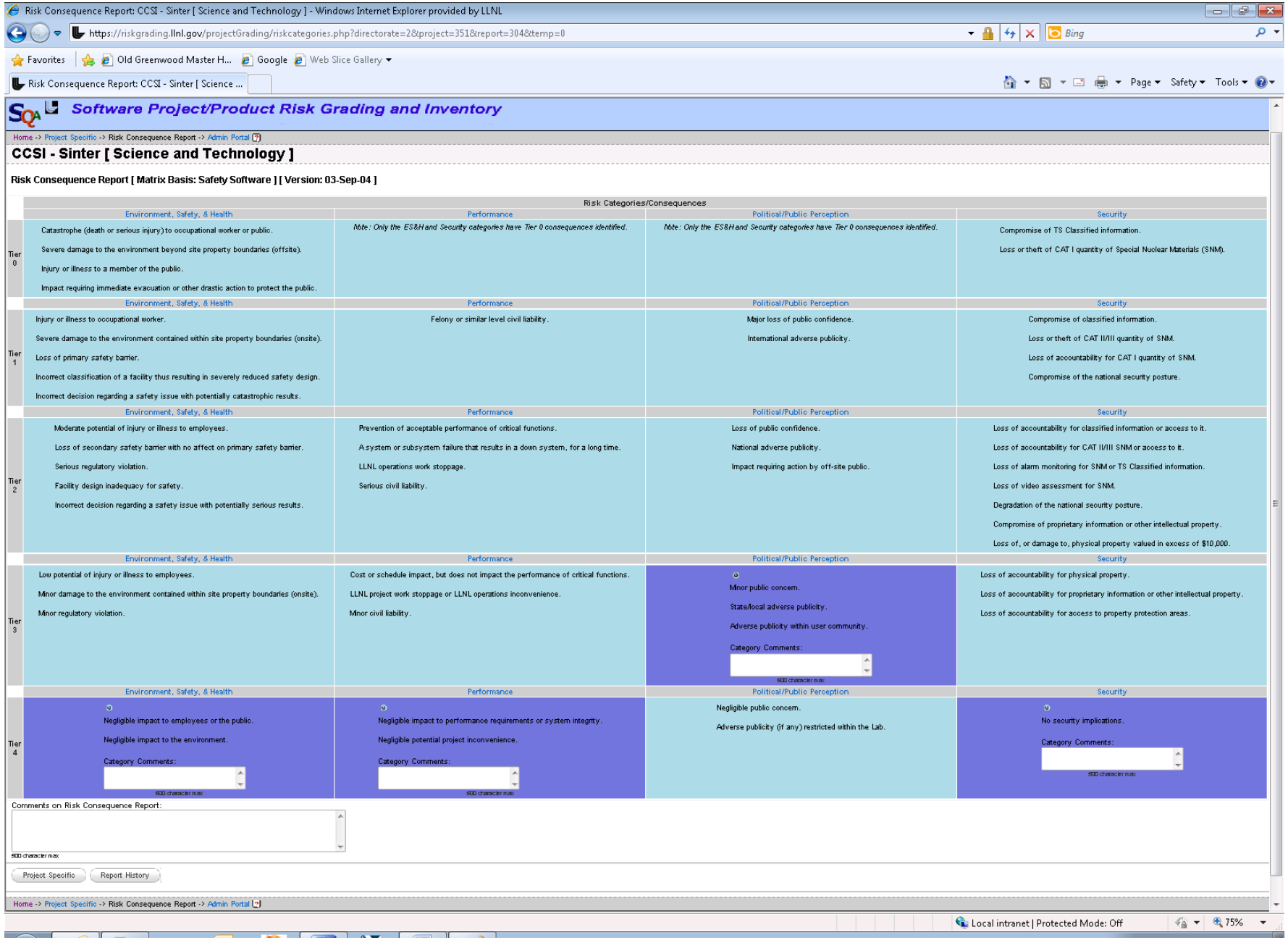
Appendix B Risk Consequence Tiers

This appendix documents the answers to questions used to determine the risk grading of the Waste codes. The risk grading is explained in the CCSI SQAP. The first step in Risk Grading is to answer the questions in the Risk Consequence Severity Table. There are four categories listed in the table below. The score is the highest tier for all categories. The second step is to answer the questions about the risk due to software environment. The two scores from each question set are used to determine the risk level.

Risk Consequence Severity Category Definitions

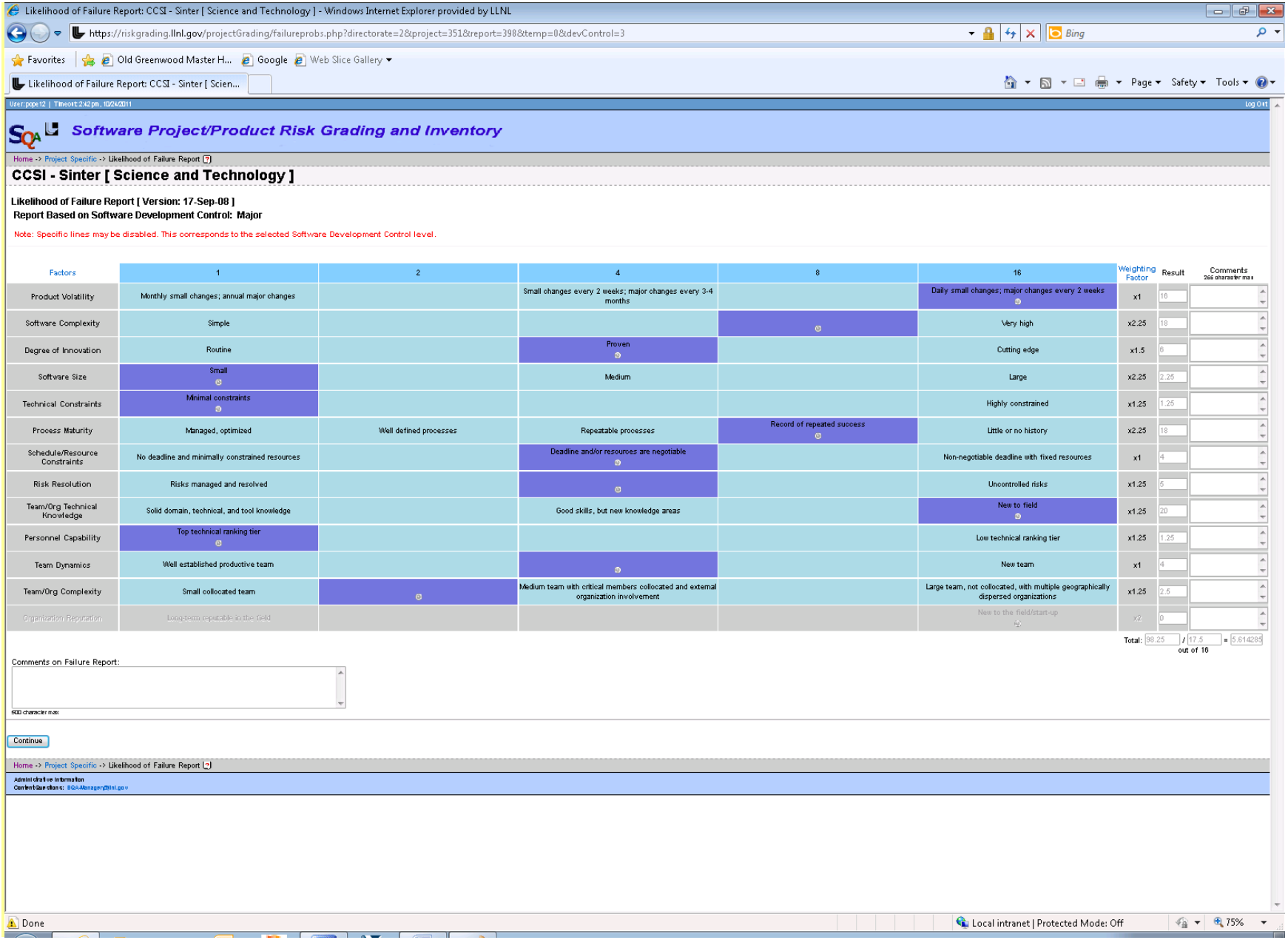
Risk Consequence Category	Description
Environment, Safety & Health	Risks to the operating and external environment, including: toxic release and cleanup. Risks to life and limb. Risks of regulatory liability.
Performance	Risks to meeting program requirements/goals. Risks of system downtime and work stoppage. Risks to the acceptable performance of critical functions, including civil liability. “Critical functions” are those important to the operation of the system or subsystem.
Political & Public Perception	Risks to governmental and public confidence and concerns.
Security	Risks to program, product, and material security.

B.1 CCSI Risk Consequence Severity



B.2 CCSI Risk Due To Software Environment

CCSI Risk Due To Software Environment Rating: _____5.61____



B.3 Risk Level

With the Risk Consequence Severity Tier and the Risk Due To Software Environment Weighted Factor identified, use the table below to determine the Risk Level:

Risk Level Assessment CCSI RL 4

Matrix Grade

[Version: 27-Aug-08] **Matrix Grade**

[Version: 27-Aug-08]

Tier 0	RL1	RL1	RL1
Tier 1	RL3	RL2	RL2
Tier 2	RL4	RL3	RL3
Tier 3	RL4	**RL4**	RL3
Tier 4	RL4	RL4	RL4
Likelihood of Failure Rating:	**0-2**	**2-8**	**8-16**

Appendix C Good Practices for Modeling Tools and Data

Traceability- References to source materials used in the model can be helpful to list. Where did the assumptions, formulas, engineering units, and data in the model come from? Consider using comment lines in the code to annotate source code or input used.

Validation - Are the sources of data values or equations the best possible source? Provide comments why the particular source data or equations were used and any certifications or pedigree that would be useful for users to be aware of. Beware of errors in text book code, compile and test text code before using before using.

Cloning - Does all data emanate from the same controlled source (for instance a data base or repository), or do individual models contain definitions locally. Good practice is to use a universal source of modeling data which is version controlled and traceable so assumptions will all be the same between various models. Also, changing or revising data can be accomplished in a central location, not requiring changes in numerous separate locations. Comments can reference the source and version used. All users of the centralized data should be notified when a change is made.

Versioning - Is the version of the modeling tool used still supported and available? If not is the newer version backwards compatible? This information can be inserted into comments. This applies to versions of compilers, options used in the compiler, libraries, operating systems, and other model support codes.

Verification - Are definitions and engineering units all defined the same way? Provide comments which describe why particular units of measure are chosen and what their values and engineering units are.

SQA - Is Software Quality Engineering applied uniformly for modelers or left up to each modeling effort?

Risk - Is a uniform risk grading practice followed and what were the results of the risk grading?

Modularity / Data Hiding - Can data or equations used by a model be changed by anyone or are permissions controlled? Source code and data should have access control with appropriate levels of permissions.

Version Control - Each version should be assigned a unique number, the reason for the version change should be noted as well as who made the change, time and date of the change. Both of these can be accomplished using a code repository or document management tool such as Team Forge or SharePoint.

Version Control - Can a diff be done on code to show exactly what has changed between versions? This can be accomplished in most versioning tools for model code and data.

Version Compatibility - Has the model been developed in one version of the modeling software but is now being used on a different version of the modeling software? Compatibility tests should be noted in comments to allow users to understand on which versions of the modeling tools the answers have been verified on.

Regression Testing - If appropriate has the new version of the model been demonstrated to not break when using previous model input and that the new version of the model software is backwards compatible with previous models.

V&V - Any Verification and Validation of values, formulas, answers, engineering units, or other items should be noted in comments, who performed the V&V and when.

Maturity - Information about how the model was developed, how long has it been in use, how many people use the model (if known) should be noted and updated as required in comments.

Accuracy - Information about the accuracy of the model or uncertainties in the model should be noted in the comments.

Repeatability - Can the model produce the same outputs given the same inputs over time?

Testing - Does the model contain test cases of check data with expected results to help assure the model is properly working?

Performance - Is the amount of time used for the model to reach an expected result reasonable and consistent?

Documentation - Is the procedural language and data used in the modeling software well commented? Is a user guide provided with the modeling software to aid the user?

Overwriting - If multiple users can collaborate on the model is a check in / checkout capability provided to not allow overwriting the same section of code, code repositories and Team Forge and SharePoint are examples of tools that automate this process.

Write Protection - Can any user change the source code or data if it is distributed? If so how is this controlled and tested?

Disaster Recovery - If the modeling software and models were erased or destroyed is there a backup copy available off site?

Appendix D Good Practices for Spreadsheet Models

Archiving - Consider archiving Excel files based on changes on some reasonable basis. For example, archive and maintain the data sets at the end of the day and use a unique filename or identifier for that data set or file. Tools such as Team Forge and Share Point help automate this process. Another approach would be to export data from the spreadsheets (both the data inputs and calculated values) into a database for data archive purposes.

Permissions - Access to the spreadsheet rows, columns, data, formulas, and macros should be limited to certain users so that unintentional model errors cannot be introduced. This would limit the potential for users to inadvertently change the formulas or data and thereby generate erroneous data

Accessibility – Limit accessibility to only those with a need, and the knowledge, to make changes

Input Control - Use of drop-down or check box menus for data entry should be used whenever possible to minimize topographical errors.

Readability - Use “named values” for cells in equations within Excel instead of cell references when possible. This makes verification of data calculations and data flow much easier.

Version Control - Note the version of Excel in which the spreadsheets are used, and note on what Operating System (OS) the version of Excel is used, and platform type (32 bit or 64 bit).

Commenting - If there are Visual Basic (VB) macros their purpose and a description of what they do should be in the comments (other than within the code of the macro itself or the name of the macro).

Modularity - Breaking down equations into smaller intermediate results helps to assure correct calculations.

Correctness - Assure correct order of execution and precedence of equations.

Typos - Caution on cut and paste from title bar formulas inadvertently inserting cell numbers

Testability - Consider implementing error checking within the spreadsheet itself, having a few problems with known answers that can be run after changes are made.

Version Control - Can a diff be done on the spreadsheet to show exactly what has changed between versions? This can be accomplished using diff tools for spreadsheets such as ExcelDiff. Using the track changes feature on spreadsheets also can show changes over time to a single spreadsheet, who made the change and why.