DOE Energy Facility Contractors Group
Software Quality Assurance Task Group
Reference Document EFCOG35.01.00-2010

Software Quality Engineering Guidelines for the Development and Support of Software Supporting Research

October 2010

Prepared by the

Energy Facility Contractors Group
Integrated Safety Management & QA Working Group
Quality Assurance Subgroup
Software QA Task Group

Blank Page

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof or any of their contractors or subcontractors.

This report will be reviewed and updated on a periodic basis. If you have any corrections, additions, or deletions, please contact the Quality Manager at your site for the name of a contact on the Energy Facility Contractors Group, Integrated Safety Management & QA Working Group, Quality Assurance Subgroup, Software QA Task Group.

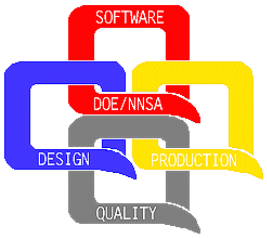
Blank Page

Software Quality Engineering Guidelines

for

The Development and Support of Software Supporting Research

October 2010



Abstract

This white paper provides information and guidance to the Department of Energy sites on software quality and software quality assurance related to software that is used to support research efforts.

Acknowledgments

The Software QA Task Group of the *Energy Facility Contractors Group, Integrated Safety Management & QA Working Group, Quality Assurance Subgroup,* initiated Work Item #35 to develop a quality report addressing the application of Software Quality Assurance to software used to support research efforts. The Software QA Task Group members and other major contributors are listed below:

Sid Ailes	EnergySolutions Task Group Chair
Dennis Adams	SAIC
Maria Armendariz	SNL
Mike Elliot	AWE
Cherri French	NTS
Barbara Campbell	LLNL
Frank Campbell	SRS
Thomas K Larson	INL
Patricia Loo	INL
Debbie Mavros	NTS
Molly Minana	SNL
Frank Olivas	KCP
Steve Painter	LANL
Dave Peercy	SNL
Debbie Rosa	Y12
Ron Schrotke	PNL
Anton Tran	DOE-ABQ
Roger Ward	PX
Mike Williamson	SNL
Scott Matthews (Chair)	LANL

This Working Group wishes to acknowledge ANSI ASQ Z1.13-1999, "American National Standard, *Quality Guidelines for Research"* as the basis for this work product.

Table of Contents

1.0 Introduction and Scope 1
1.1 Background 1
1.2 Purpose and Scope of Document 1
1.3 Objectives 2
1.4 Risk-Informed Concepts 2
1.5 Software Quality Engineering Concepts for SSR 3
1.6 General Approach for Software Quality Engineering (SQE) for SSR 8
2.0 References 8
3.0 Management of Software Quality Engineering for SSR 9
3.1 General Principles 9
3.2 General Software Quality Planning 9
3.3 Risk Analysis and Risk-Informed Graded Approach 10
3.4 Configuration Management 11
3.5 Requirements Management 13
3.6 Design and Implementation 13
3.7 V&V and Testing 13
3.8 Maintenance of Software Supporting Research 15
4.0 Research Plan for the Development and Maintenance of SSR 15
Appendix A: Template-Research Plan for the Development and Maintenance of SSR 16
Appendix B: Definitions 23

1.0 Introduction and Scope

1.1 Background

Dr. Werner von Braun stated "*Research is what I'm doing when I don't know what I'm doing."* Dr. von Braun's insightful statement still rings true and thus poses a dilemma: How can a researcher be assured of meaningful and reproducible research results if the eventual "goal" may be unknown and the path to the goal is strewn with complexities?

A research scientist was queried about a computational result that was published in a respected journal. Upon reflection of possible accuracy concerns, the researcher decided to repeat the computations and check the analysis. Alas, the specific software version used to support the analysis could not be reproduced. Hence, the professional query that called into question the computations could not be specifically verified or refuted. The software configuration management, verification and validation practices, and more generally the rigor of the researcher's research results also were called into question. Unfortunately, this is more common than might be expected. Software that supports a research effort can be important to the validity of the research results and, hence, should have sufficient and appropriate software assurance rigor to undergird the research.

As placing a fulcrum at a correct point can enable large masses to be moved, leveraging quality at the "correct" places within a software development or modification life cycle can positively affect the quality of research results that depend upon the use of the software. Quality is not an abstract concept but rather the use of well-vetted principles employed regularly within industry. Quality remains the application of one of the prime attributes of the scientific method, i.e., the use and management of appropriate controls to achieve a desired objective. As in industrial production, quality within research is vital to mitigate the possible consequences of failure or defects occurring within the delivered research product(s).

The guidelines presented within this document support the development and support of a "research project" employing the scientific method for hardware, interface and software components using a risk-informed graded approach for scripts, customized software, acquired software, commercial-off-the-shelf software, and prototyped and developed software applications. These guidelines will enable reproducibility of the software and its algorithms thereby providing a capability to assess the pedigree of the research.

1.2 Purpose and Scope of Document

In 1999, the American Society for Quality (ASQ) issued the approved ***ANSI/ASQ Z.1.13-1999, Quality Guidelines for Research.*** This ASQ standard supports the development of a quality system for basic and applied research, whether hardware or software. However, no quality standard or guidelines presently exist to support the development of software supporting research applications for discovering the truth using the scientific method. The purpose of this document is to provide that guidance.

"Software Supporting Research (SSR)"could have minimal "pedigree" and perhaps fall under the umbrella of "Research Software" wherein "research software" is typically prototype software being developed as a deliverable for the research for "discovery" purposes. When the "research software" algorithms and objectives are better understood and more clearly defined, then the researcher might benefit by adopting the principles within this guidance document. Transitioning the "research software" to the "Software Supporting Research" paradigm with the appropriate application of quality principles can aid the researcher. For example, the transition might provide the basis for reproducing the research results, enhancing the research pedigree among peers, and providing the needed information vital for potential additional research funding and/or publication within a respected peer-reviewed journal.

SSR may be "commercial-off-the-shelf (COTS)", "government off-the-shelf", or somehow developed or acquired software that is being used for support of the research effort.

1.3 Objectives

This document can be used as a quality system model in situations where one or more of the following apply:

- An objective of the research is to use the scientific method to produce a useful product or data for publication release, operational decisions or further research proposals so the pedigree of the software must be appropriately robust and high quality;
- Software results must be reproducible and defensible to support the validity of the research results and to ensure all members of the research team have a consensus regarding the objectives of the SSR;
- Applying tailored quality activities supports rather than impedes the research;
- The software algorithms are dynamic;
- A baseline is required for evaluating an organization's capabilities for managing and performing basic and applied research.

1.4 Risk-Informed Concepts

To support the five listed objectives, an assessment of the research pedigree is necessary. This knowledge prepares the researcher to apply a "risk-informed" approach. A risk-informed approach means that the software quality activities and "work products" are directly correlated with the research pedigree, i.e., the breadth and depth of the subject knowledge base and resources already committed to the research. Basically, the software quality assurance processes and tasks discussed herein must mirror the confidence level appropriate to the research that the SSR supports.

Risk may be classified as “high”, “medium”, or “low”. In practice, if the research objective is “discovery”, the risk may be perceived as “low”, i.e., the consequences of failure are considered minimal since any unexpected or adverse research results just mean that the research models or experiments must be revised. Thus, the number and content of the work products must have sufficient and appropriate content to support reproducibility of the research results. If the SSR objective is to explore the algorithm(s) and its attributes, the risk may be considered somewhere between the low to medium gradient. However, if the SSR objective is to provide decision support for new product(s) and/or process(es), incorrect or adverse results could influence costly mistakes in product or process implementation. In summary, a “risk-informed “approach simply means the researcher should effectively use more quality assurance activities and provide more content within the tailored work products if failures, defects within the SSR could have undesired consequences.

1.5 Software Quality Engineering Concepts for SSR

Table 1 on the next page lists some of the major software quality engineering concepts for SSR with pointers to the sections within this document providing recommendations using a risk-guided approach for SSR.

Table 1:FAQ for SSR

Question Regarding SSR	Answer	SQE Mitigation Strategy	Section(s) Within This Document
What does a "risk-informed approach" mean in practice?	A risk-informed approach is understood to mean that the researcher has some insights regarding the risks and tradeoffs if the SSR fails or provides incorrect results. Even a rudimentary risk-informed approach permits the researcher to provide focus and prioritization for the resources and work products to assure the SSR sufficiently addresses the research objectives. With some understanding of the SSR risks, the researcher can determine not only what work products might support risk mitigation but also what level of rigor and detail will support the objectives of reproducibility and common understanding of the SSR objectives.	Software Quality planning and Risk-informed Approach	3.2,3.3
What level of documentation would be useful to the researcher to maintain integrity and reproducibility of the research results?	The extent and volume of documentation for SSR should be commensurate with the risks associated with the consequences of not having documentation. Consequently, sufficient detail should exist within the documentation to enable "common understanding" and acceptance by the research peer group to ensure the objectives/assumptions/limitations are known and commonly understood. Likewise, sufficient detail must exist within the documentation to permit "reproducibility" of the results by any other researcher as well as providing sufficient materials to document the research and results within peer-reviewed journals.	Risk-informed Approach	3.3
How can the researcher work with a team on a research project and assure that the quality activities are appropriately addressed?	The researcher must sufficiently communicate with and to the team supporting the research to ensure common understanding. Additionally, the lead researcher should strive to establish the infrastructure to encourage creativity and "out-of-the-box" thinking. In practice this means that typical time-consuming meetings and reporting responsibilities are minimized while encouraging creative thinking. However, collaborative tools such as "Wiki" pages should be encouraged to ensure that ideas and brainstorming results are captured. This question is also related to the following question on the next page.	Software quality planning	3.2

Question Regarding SSR	Answer	SQE Mitigation Strategy	Section(s) Within This Document
How can the integrity and reproducibility of any aspect of the SSR be assured?	Both the integrity and reproducibility of the SSR are directly correlated with configuration management and Verification and Validation (V&V). Configuration management is fundamental to ensure the integrity and reproducibility of any aspect of the SSR. This means that all software and documentation be assigned a unique identifier for the type of item along with the date and revision information. This enables the researcher to return to a previous "working" version when a change introduces undesirable results. It also enables the researcher to reproduce previous results – even long after the original results were produced. V&V also supports the integrity and reproducibility of the SSR with appropriate risk-informed reviews and testing to ensure the SSR was developed correctly per the objectives and requirements and functions as intended.	Configuration Management, and Verification and Validation (V&V) and Testing	3.4 & 3.7
If the researcher wants to purchase the SSR software, how does the researcher decide whether to purchase the software or not?	A key component of determining whether to purchase the SSR or not must be based upon the software requirements for the SSR. After a determination of the software requirements, the researcher should visit with the vendor to determine if the vendor's software application does indeed satisfy the SSR requirements. Part of this process requires the researcher to not only understand the software application's pedigree but also the domain of use. Knowledge of the application software's pedigree will enable the researcher to not only understand what in-house testing the vendor may have done but also how the software has benefitted other users. Information regarding the domain of use will help guide the researcher to understand whether the software's operational parameters and algorithms are *a propos* for the intended SSR application and whether the application's functionality satisfies the required functionality and domain of use.	Requirements Management, Verification and Validation (V&V) and Testing.	3.5 & 3.7

Question Regarding SSR	Answer	SQE Mitigation Strategy	Section(s) Within This Document
How can the researcher best decide whether software tools might be useful? If software tools may be useful, how can the researcher best determine which software tools to choose and use?	This question is related to the question above; the provided answer has some application for the decision about acquiring software tools. If the SSR is rather large, i.e., a few thousand lines of source code, then various testing tools might be beneficial. If the application is fairly small, then simple test cases and insertion of well-placed "print" statements can provide guidance regarding the behavior of the algorithms. Configuration management software is always recommended to ensure software and test case/data integrity and reproducibility. A risk-informed approach to configuration management is readily acknowledged to be reasonable.	Configuration Management, Testing	3.4 & 3.7
How can the researcher begin to develop software requirements?	Document the expected objective. This is basic to the scientific method. As Claude Levi Strauss stated "T*he scientific mind does not so much provide the right answers as ask the right questions."* Documenting the questions to be addressed by the research is vital for common understanding, reproducibility and an assurance that the correct questions are being asked. By documenting the questions and objectives, the researcher also has the opportunity to ruminate and determine if the research and the SSR has the proper focus and whether the objective is appropriate and what other alternative outcomes might exist.	Requirements Management	3.5
How can the researcher best decide upon algorithm applicability?	Prior "homework" is the foundation for deciding upon algorithms. The researcher should perform a literature search and/or "data mining" techniques to investigate the limitations, assumptions and/or constraints of various algorithms. Hand calculations or comparison of the results against accepted and validated algorithms also are acknowledged to be acceptable and appropriate practices. Prototypes and computational steering also will help the researcher understand and define the algorithms to support the SSR.	Requirements Management	3.5
How can the researcher successfully design and implement SSR?	Since the requirements and objectives may be fluid, adoption of software engineering methods such as prototyping or agile development methods are recommended. A common approach is "trial-and-error" methods such as used by Thomas A. Edison. It should be remembered that even though Edison practiced "trial-and-error", he documented what he had learned from each trial, such as all the pro and con attributes and limitations for each "trial" before proceeding.	Prototyping and Agile Development	3.6

Question Regarding SSR	Answer	SQE Mitigation Strategy	Section(s) Within This Document
How can the researcher be confident that the results are reasonable and as intended?	This question and response is similar to the above question regarding the "algorithm applicability". Both prior "homework", i.e., literature search and/or data mining techniques can provide guidance about the application and limitations of any algorithms developed and integrated within the SSR. Comparisons of results with either "actual" experimental results and/or other accepted and validated software applications are another means to develop confidence in the SSR results.	Verification and Validation (V&V)	3.7
How should V&V be applied for SSR?	Verification and validation processes provide increased confidence to the researcher that he/she is progressing toward discovery and new knowledge (or differently applied understanding/knowledge of other research domains).	Verification and Validation (V&V)	3.7
How should rework and corrections be performed for SSR?	Any rework and/or corrections that must be made for either the SSR itself or supporting documentation, both the date and rationale for rework or corrections must be documented. Both the "pre-revised/pre-reworked" SSR and/or supporting documentation **and** the revised/reworked SSR and documentation must be retained in case the perceived defect or need for rework either change or were incorrectly implemented. Keeping the older version will enable the researcher to "fall back" to the earlier functioning SSR (or useful document) if necessary. In addition, if the older versions are retained then later analyses might reveal either common cause issues, more effective algorithms, or better clarity.	Configuration Management, Maintenance	3.4 & 3.8

1.6 General Approach for Software Quality Engineering (SQE) for SSR

To address the questions within Table 1 above, the application of a more disciplined approach to the SSR is believed to enable a better understanding of the research problem domain, which in turn, will result in fewer defects. The particularly vital software quality engineering processes to invoke for SSR are an assurance that the perceived problem is being correctly addressed; and the integrity and reproducibility of all the SSR work products and results are easily replicated. Hence, this guide is intended to provide recommendations for developing a "Research Plan" that describes the software quality engineering principles required for the integration of requirements management, configuration management and verification and validation into the scientific method for SSR while acknowledging reasonable trade-offs based upon a risk-guided approach to managing the SSR.

Refer to the ***ANSI/ASQ Z.1.13-1999, Quality Guidelines for Research,*** for overall quality guidance for research.

2.0 References

1. ANSI/ASQ Z1.13-1999, American National Standard: Quality Guidelines for Research
2. CMMI: Guidelines for Process Integration and Product Improvement, Mary Beth Chrissis, Mike Konrad, Sandy Shrum, Addison-Wesley, July, 2004
3. Computational Steering, Robert van Liere, Jurriaan D. Mulder a Jarke J. van Wijk, Center for Mathematics and Computer Science, Amsterdam, Netherlands, March, 1998
4. Refining Feature Driven Development – A Methodology for Early Aspects, Jianxiong Pang Lynne Blair, Computing Department, Lancaster University, Lancaster UK, 2004
5. IEEE 610.12-1990, IEEE standard glossary of software engineering terminology
6. Merriam-Webster Online Dictionary, 2009
7. PC Magazine Encyclopedia, Ziff-Davis Media, 1996-2009
8. Quality Report SQAS93-003, "Preferred Practices for Software Quality within the Nuclear Weapons Complex", August 1993
9. Quality Report SQAS96-003, "Software Quality: A Guide to Responsibilities and Resources", December 1996
10. Quality Report SQAS97-001, "Guidelines for Software Measurement", April 1997
11. Quality Report SQAS-031, "Applying Agile Methods to Weapons/Weapons Related Software", October 2005
12. Software Engineering: A Practitioner's Approach, 6th edition, Roger Pressman, McGraw-Hill, 2005
13. Software Testing Techniques, Second Edition, Boris Beizer, Van Norstrand Rheinhold, 1990
14. Test Driven Development: By Example, Kent Beck, Addison-Wesley, 2003
15. Verification and Validation of Simulation Models, from "Proceedings of the 1998 Winter Simulation Conference", Robert G Sargent, Simulation Research Group, College of Engineering and Computer Science, Syracuse University
16. Writing Effective Use Cases, 1st edition, Alistar Cockburn, Addison-Wesley, January 2000

3.0 Management of Software Quality Engineering for SSR

3.1 General Principles

As stated within ***ANSI/ASQ Z.1.13-1999,*** the "research organization should require scientists to develop a research plan or research proposal that describes the conduct of the research".[1] This research plan should describe the general guidelines for applying software engineering principles using risk guidance as well as delineating the various responsibilities of the staff supporting the development of the software supporting the research.

3.2 General Software Quality Planning

As in other scientific domains, planning the research is vital for success. This planning must explicitly and implicitly address several areas pertinent to software supporting research. Part of the problem definition must include ***what will be addressed*** by the SSR and ***what will not be addressed.*** Early problem definition and bounding the problem in this manner will greatly aid the researcher to ensure proper focus and use of the limited resources.

If several people, i.e., a "team", are involved in the SSR effort, the organization and roles must be defined, but defined in such a way as not to stifle creativity. If there are resource constraints such as financial or people, the team must understand the constraints and where the focus must be. As in other areas of SSR, a risk-informed approach (See Section 4.3) should be used as the foundation and basis for quality planning.

The keystone for software functioning as intended while not performing any unintended functions is the effective use of configuration management and requirements management. As in other quality planning areas, a risk-informed approach helps the researcher to focus the planning and resources upon the topics of most importance while consciously reducing the topics of lesser importance. The topics of greater importance for SSR undoubtedly include a determination of:

- what problem will be addressed and what will not be addressed by the SSR;
- whether to develop or acquire the SSR;
- how configuration management will be addressed; and
- where and how to obtain an understanding of the software requirements for the SSR.

The researcher should make conscious decisions about the work products and the trade-offs of invoking software quality processes and developing any supporting software quality related work products.

A quick perusal of the Internet will yield useful information about "best business practices" that have been well vetted by other researchers and practitioners. A number of Web sites also have useful links about software quality or software engineering practices. Carnegie-Mellon's "Software Engineering Institute (SEI)" also has a number of papers available that discuss good software engineering practices.

The following sections of this document are intended to provide general guidance for the researcher developing and maintaining SSR. Appropriate guidance for tailoring all the software

[1] Paragraph 4, "The Management of Basic and Applied Research Projects", ANSI/ASQ Z.1.13-1999

life cycle processes and work products is provided within the next section, Section 4.3, Risk Analysis and Risk informed Graded Approach.

3.3 Risk Analysis and Risk-Informed Graded Approach

Using risk analysis, the researcher can have higher confidence that "no surprises" will occur. Risk analysis is a tool for the researcher to methodically assess any potential vulnerabilities, threats, or impacts to provide a knowledge basis for reasonable judgments.

A *caveat* regarding this risk analysis is that risks are commonly understood to be a function of a specific time span. Over a several year duration, the risks are obviously different than they would be versus the period of a few weeks or months. Thus, the researcher may need to define the time duration for any perceived risks and how the risks may change as a function of time.

The crux of risk analysis[2] is to identify the potential vulnerabilities, i.e., adverse events or resulting consequences, and a determination made of the likelihood of the adverse event occurring. Typically there are risk-tradeoffs. For example the consequences of a software algorithm defect impacting the results of a specific research experiment might completely invalidate the research results. The likelihood of the software having such a defect will depend on the level of assurance that the software practices provide. The researcher should develop a matrix of the potential vulnerabilities and/or adverse consequences versus the likelihood of occurrence. Since the "likelihood", i.e., probability of occurrence of each adverse event, is unknown, relative likelihoods are typically used. Table 2 provides an example of general risk guidance that the researcher might be able to use to support decisions and work product type and content using a risk-guided approach. The actual table might vary depending on the research project and the level of dependence on software. Depending on the resulting risk level, the researcher may choose to require a particular approach to the software quality assurance practices.

[2] Risk analysis is a determination of the likelihood of an event and the consequences if the event did occur.

Table 2: Likelihood versus Consequences Risk Table

Consequence / Likelihood	Serious to Catastrophic Consequences	High Consequences	Moderate Consequences	Marginal to Minor Consequences	Negligible Consequences
Almost Certain	1 - Highest Risk	1- Highest Risk	2 - High Risk	3 - Moderate Risk	4 - Low Risk
Likely	1 - Highest Risk	2 - High Risk	2 - High Risk	3 - Moderate Risk	4 - Low Risk
Possible/Probable	2 - High Risk	2 - High Risk	3 - Moderate Risk	4 - Low Risk	5 - Negligible Risk
Unlikely	3 - Moderate Risk	3 - Moderate Risk	4 - Low Risk	4 - Low Risk	5 - Negligible Risk

If actual probability and consequence data are available for the SSR researcher/developer, other tools and technologies such as Pareto analyses, Bayesian statistics, "fishbone" diagrams (cause-effect diagrams), fault tree analysis, etc., might further guide the researcher where a SSR failure or adverse outcomes might result in irreparable damage to the research and SSR efforts.

3.4 Configuration Management

Configuration management is probably one of the simplest quality assuring processes to invoke and yet one of the most neglected disciplines within software quality because it is often seen as unnecessary or the project seems too simple to need "configuration management".

Before a researcher reads the above paragraph about risk management, an alarm about too much rigor and process being imposed, a *caveat* is required. As in other software quality assuring processes for the SSR, a risk-informed configuration management process wherein less formal configuration management is employed is undoubtedly sufficient for lower risk and/or less complex SSR. Backing up source files and documentation files for lower risk SSR applications will provide sufficient configuration management. The researcher should at least "log" how to access the various versions and maintain a date and time stamp with the rationale for creating/modifying each software component or document.

However, if configuration management is either haphazardly used or not used, then the researcher has no mechanism to vouch for the integrity of the SSR and its verification and validation evidence. Only via configuration management can the pedigree of the SSR be certified and enable other researchers to reproduce the results and even extend and enhance the application and its algorithms.

Configuration management may seem arcane to researchers unfamiliar with the configuration management discipline. Configuration management includes four separate disciplines: (1) configuration identification; (2) configuration control; (3) status accounting; and (4) physical and functional configuration audits.

Addressing the configuration management disciplines in reverse order, configuration audits are conducted for two purposes. The first purpose is to determine if the operational (deployed) SSR's functionality conforms to its documented "behavior", i.e., does the SSR satisfy all its functional, performance and quality requirements? Secondly, configuration audits are the mechanism to provide a "baseline" definition of the list of documentation, hardware and software components that comprise the SSR. Only by listing all the individual components, whether documentation, hardware, operating system, software libraries, language compilers, and the SSR application itself, can the reproducibility of the SSR be assured.

Status accounting provides the mechanism to ascertain what SSR components have been uniquely identified and what the status is of any proposed change, i.e., whether a proposed change has been successfully implemented, is being investigated for viability or rejected.

Change management (aka "configuration control") is another important piece of the configuration management puzzle. Uncontrolled changes have unfortunately led to many hardware and software failures when a proper application of configuration management to control the evolution of the hardware or software design and application could have prevented the system failures. "Change management" in this context simply means the researcher would assess the impact of any proposed code change to be able to ascertain if the change could adversely impact another area of the SSR application. In addition, the researcher may need to prioritize any change to the SSR, especially if any changes could potentially have adverse consequences in another area of the application. Only via the proper use of change control can the researcher vouch for the pedigree and integrity of the SSR application, its ancillary supporting documentation and the software output results.

Uniquely identifying all the work products, i.e., the documents, all software, and test data, will greatly aid the researcher to ensure reproducibility either later by the researcher or by other people who may support the research and SSR at a later date. Typically, the identifier may look something like this:

Type_of_product_date-time_version.

where "Type_of_product" may be a unique identifier such as "**SRS"** (software requirements specification), or "**XYZ_SC"** for a particular software, where **"XYZ"** is a name for a particular piece of software, and **"SC"** is for source code. A date/time stamp should be included as well as a version and variant number.

In summary, there are a number of very good and well accepted configuration management software tools and publications available to guide a researcher. Some are downloadable freeware and others may require a small license fee. The use of configuration management software can certainly pay great dividends and prevent angst simply by enabling a documented history of the software changes while maintaining integrity of all the electronic work products supporting SSR.

As in all the other software engineering and software quality disciplines recommended within this document, software configuration management must be guided with risk-informed knowledge.

3.5 Requirements Management

Defining "what the problem is" and "what the problem is not" is a key component of research investigation and requirements management. As part of requirements management, understanding boundary conditions, algorithm applicability for the *specified* domain as well as the assumptions and constraints associated with the algorithm to be implemented will greatly assist the software requirements definition. Defining the boundary conditions also implies that the conditions and interfaces at the boundaries are known and understood for the SSR. This often is not a trivial task. Diagramming the SSR and interfaces may support this task and the possible research use cases for the SSR

The accepted "best practices" for both research and industrial applications is to understand and document the major functionality and expected results, proposed algorithms (or source to locate the algorithms), how error checking will be managed (if at all), and any constraints, assumptions and limitations. Documenting the target problem domain and assumptions for the SSR could greatly aid the researcher (or subsequent researchers) from later misapplying the SSR to unintended problem domains.

An important aspect of requirements management is to define the data, its type and length, (e.g. 32 bit versus 64 bit), and how it will be managed. Similarly, the researcher must consider the processing steps if invalid or corrupt data or calculations result, such as out-of-range data, lost or missing data, or invalid alphanumeric values are entered. Another possible concern from invalid data or pointer values occurring within either the input values or application are that these values could corrupt the valid data created up to that processing step and negate the likelihood of reproducibility. All these topics are decisions the SSR author must make and determine how much anomaly management is appropriate using a risk-informed approach.

3.6 Design and Implementation

Developing a "proof-of-concept" via creating a software application to instill confidence and assurance of an algorithm's applicability is a vital aspect of SSR. "Feature Driven Development (FDD)[3]" and/or "Test Driven Development (TDD)[4]" are useful prototyping techniques to gradually and incrementally design and develop SSR. By gradually and incrementally developing the SSR, the researcher can refine and narrow the domain definition and the associated parameters. Computational steering further supports this objective by being able to inject queries and plotting capabilities in the SSR. A risk-guided approach applied to design and development practically means that the applied rigor should be risk-guided for developing and documenting the SSR. Trial and error may be as appropriate for lower risk applications as much as computational steering may be an appropriate development mechanism for a more complex, higher risk informed SSR application.

3.7 V&V and Testing

If the research develops the SSR versus purchasing the application and possibly customizing the application for the particular SSR objective, some variations in the approach to testing the SSR may be required.

[3] See Reference 4
[4] See Reference 12

If the researcher developed the SSR application, then the researcher would need to perform unit testing and integration testing to provide the assurance and confidence that the algorithms and parameter passing is occurring as intended. Neither unit testing nor integration testing would be required if the researcher purchased the application for SSR purposes. In the case of purchasing the application or possibly modifying an application for the target SSR objective, all testing should focus upon rigorous testing of the algorithms and application domain to ensure that the software behaves as anticipated within the prescribed and defined operational domain.

Since the researcher may be working with complex unknowns, nothing should be assumed about nominal values or "expected" results for the SSR. In practice, many times the researcher may not know what results to even expect for SSR results; if the results totally diverge from the expected output data range, the researcher must determine whether the algorithm application, input data, invalid assumptions or an incorrect understanding are culpable

If possible and appropriate using a risk-informed approach, the test cases should include the data boundary conditions or out-of-range conditions. The researcher should be especially mindful of the old adage[5] "program testing can be used to show the presence of bugs, but never to show their absence."

Various software tools are available for the researcher to support the software testing effort and are recommended to support the objective of a correct and robust SSR. Such software tools as "static analyzers", i.e., applications that analyze the application for modular construction, "memory leaks" and language syntax, are basic software tools. Other software tools are coverage analyzers and test key capture applications. Coverage analyzers can be used to determine if all the looping and conditional constructs have been tested. Other testing tools can capture the key strokes of the researcher and then be edited later and "played" back, i.e., re-executed with the researcher's test modifications thereby relieving complete re-entry of all the test data repeatedly.

Verification and validation are vital to the researcher to ensure the researcher is addressing the intended problem with the correct and appropriate algorithms. Comparing the SSR results against hand calculations, comparable code results or a good suite of test cases are all acceptable for providing confidence in the SSR and its results.

Many standards exist to provide guidance for conducting research or analyzing the experimental results. The American Society for Quality (ASQ) has many useful analytical tools and standards to support the analyses of SSR results. Other professional and industry organizations publish standards that may support the V&V for the SSR. Various branches of the Federal and State governments also produce standards. Web searches also could yield links to supplement and complement the existing quality and analysis tools and standards.

Response surface[6] methods could further support the V&V efforts by providing sensitivity and uncertainty analyses of the SSR algorithm(s).

[5] Edsger W Djikstra, 1970
[6] "Response Surface Methodology", CASO Technical Report, Kathleen M. Carley, Natalia Y. Kamneva, Jeff Reminga October 2004, CMU-ISRI-04-136, Carnegie Mellon University School of Computer Science

3.8 Maintenance of Software Supporting Research

Maintenance of the SSR may be due to any or all of the following three reasons:

1. a defect was discovered in the SSR, requiring a correction;
2. a change is required to improve the SSR's performance, functionality, or maintainability;
3. a modification is necessary because of the need to adapt the SSR to a different operating system, a new CPU, or some other environmental modification of the SSR operational environment.

When any of the above conditions occur, the SSR will need software maintenance. Unfortunately, as software is developed, it typically contains 6 to 30 defects for every thousand lines of code[7]. Defects can obviously impact whether the results are correct or whether the software even functions as desired. Removing the defects can be frustrating and tedious, requiring patience from the researcher. The size of the SSR application, the experience of the researcher developing software applications, the complexity of the algorithm and the carefulness of the researcher to study and review the application as it is developed all contribute to the number of defects endemic in the SSR. Software tools, independent reviews and old-fashioned judiciously placed "print" statements within the SSR can assist the researcher to gain insights as to the nature of the defect.

4.0 Research Plan for the Development and Maintenance of SSR

ASQ recommends the content for a "Research Plan" within Sections 4 and 5 of the ANSI/ASQ Z1.13-1999, *"Quality Guidelines for Research".* Appendix A contains the template format for the Research Plan for the Development and Maintenance of Software Supporting Research.

[7] Page 34, "50 Years of Software: Key Principles of Quality", James A. Whitaker and Jeffrey M. Voas, IEEE IT Pro, November/December 2002

Appendix A: Template-Research Plan for the Development and Maintenance of SSR

Research Plan for the Development and Maintenance of Software Supporting Research

ASQ recommends the content for a "Research Plan" within Sections 4 and 5 of the ANSI/ASQ Z1.13-1999, "Quality Guidelines for Research". If SSR is an integral part of the research, the paragraphs below describe the recommended content of a "Software Supporting Research Plan" as a possible addendum to an existing "research plan." The purpose of this additional content is to address the development and maintenance of SSR and the application of appropriate software quality engineering processes and work products to assure a more robust application and fewer defects.

The researcher should be mindful of the content and detail to include within the SSR Plan using a risk-guided approach, i.e., for lower risk SSR applications, less detail is required to enable "common understanding" and "reproducibility" by another researcher. This SSR Plan is really evidence of the pedigree of the software and the research itself.

A1.0 Problem Statement (Section 1.0 of the SSR Plan)

This section will discuss the perceived problem requiring the analysis and application of software to simulate or support the solution of the identified problem and how the software is anticipated to support the research objectives. This section should also state the limitations of the solution set and potential limitations or difficulties with the approach(es) being used.

A2.0 Assumptions and Constraints (Section 2.0 of the SSR Plan)

This section should document any assumptions or constraints for developing the software, including any Federal or State laws or institutional policies that must be addressed in the development or maintenance of the software.

This section also must address and caveat the limitations and applicability of this software application to other research problem domains and/or the incorrect use or misuse of the software application to other research areas.

A3.0 References (Section 3.0 of the SSR Plan)

This section should include any applicable technical references including journal articles, laboratory notes, textbooks or laws or policies noted within Section 2.0 of the SSR Plan.

A4.0 Acronyms & Definitions (Section 4.0 of the SSR Plan)

This section documents any acronyms, definitions or specialized terminology required to understand or apply the software to the research problem domain.

A5.0 Planning (Section 5.0 of the Research Plan)

A5.1 General (*Section 5.1*)

If this software supporting research is related to other research either previously or currently being done either institutionally or globally, this subsection should address the objective(s) of this particular SSR application. The similarities and/or variances from either previous SSR versions or other similar applications should also be included within this section of the SSR Plan.

A5.2 Quality Organization (*Section 5.2*)

If any institutional quality or software quality organization already exists, this subsection should state how the quality organization could support the development and maintenance of any software supporting the research always keeping in mind the appropriate risk-guided processes and work products.

If an organizational infrastructure already exists to support quality assurance, the researcher should leverage the existing expertise and resources and so state within the SSR Plan.

A5.3 Roles and Responsibilities (*Section 5.3*)

This subsection should specify who is responsible for the research and who is responsible for developing or managing the software supporting the research and any required reporting or communication paths about the status of the software.

A5.4 Schedule and Deliverables (*Section 5.4*)

If there is any schedule or required formal reports or software documentation, this subsection should specify the schedule or deliverables (and possible template guidance).

A5.5 Resources (*Section 5.5*)

This subsection should document any required resources to support the software development or maintenance, including staff, computer resources, computing peripheral devices, compilers, software libraries, and experimental data.

A6.0 Risk Management (Section 6.0 of the SSR Plan)

This subsection must address any potential risks and the appropriate risk-guided approach(es) to monitor and mitigate the risks resulting from incorrect use or adverse conditions resulting from adverse software functionality.

Since this SSR Plan is to address the application of software quality engineering processes, appropriate software quality engineering processes and work products should be implemented based upon a risk-guided and tailored approach and be documented within this section of the SSR Plan.

A7.0 Configuration Management (Section 7.0 of the SSR Plan)

A7.1 Configuration Management (*Section 7.1*)

This section should describe how "controls" will be integrated into the research process whether these controls are "control" management of the SSR itself as well as any software results that support the research. These controls are implemented within the software as configuration management but complement the scientific method and its objectives. What work products will be created and how and why they will be retained and for what time period must be documented within this section.

Configuration control is the configuration management discipline that enables reproducibility. By being able to track all the changes and the dates and times the changes were made permit an accurate accounting of any work product, including the software itself. Consequently, as either documents or software applications are developed, any version can reconstructed and earlier versions can be reconstructed if a later version is in error to enable a "roll-back" to a previous working version.

As noted in previous discussions of the SSR Plan, the caveat should be noted that all discussions within the SSR about configuration management should be based upon a risk-guided approach.

A7.2 Baseline (*Section 7.2*)

This section will document how the operational baseline will be derived and documented to ensure reproducibility of the software supporting the research. This operational baseline documentation includes the hardware and supporting peripherals, operating system, compilers and compiler flags, software libraries, source and object code, executable(s), documentation, regression test suites, test results, and all supporting documentation.

A7.3 Data and Document Retention (*Section 7.3*)

This subsection will document what software entities will be retained, where they will be retained and for how long. This is required to ensure reproducibility in the eventuality of corruption or loss of the software, data or results.

A8.0 Requirements Process *(*Section 8.0 of the SSR Plan)

A8.1 General Approach for Defining and Understanding Requirements (*Section 8.1*)

This subsection should describe how the requirements will be determined and defined.

A8.2 Source of Primary Algorithm(s) (*Section 8.2*)

This subsection should document how the algorithm(s) to be defined and implemented within the software application will be derived. If the algorithm(s) are documented within any reports, laboratory notes, or journal articles, this subsection should state the sources.

If the algorithm(s) are derived by the researcher via trial-and-error or domain knowledge, the assumptions/constraints/domain limitations of the implemented algorithm should also be documented.

A8.3 The Use of Acquired, COTS or Customized for SSR (*Section 8.3*)

If either acquired or COTS software will be used to support the research, the researcher must ensure that the applicable parameter domain and algorithm limitations are well understood and defined before using the acquired or COTS software for the SSR. In addition, the researcher should try to understand the software's pedigree to ensure the application has been well tested and will function as intended while not producing any unintended adverse functionality. All the known operational conditions and limitations discovered via trial-and-error and use of the SSR should also be included within the SSR Plan.

Similarly as with acquired or COTS software, the same type of understanding as for acquired or COTS software must apply to "customized" software. Since "customized" implies that either acquired or COTS were modified for a particular need, the researcher should document the rationale, assumptions and known limitations when using the COTS/acquired software for the SSR.

As previously stated, the assumption is that the researcher should appropriately apply a risk-guided approach when documenting the use of COTS/acquired software for SSR.

A8.4 Software Quality Engineering Support for Requirements Development (*Section 8.4*)

This subsection should document how the algorithms and requirements will be verified for correctness, feasibility, verifiability, unambiguousness, and completeness before implementation in the software application. Reviews and/or checklists could be used to support the accepted software quality engineering practices.

This subsection should emphasize the application of the scientific method, if possible. For example, the following information should be included:

- *Defining the question(s) to investigate (with a tacit understanding of what **is** being addressed and what **is not** being addressed)*
- *Gather information and resources*
- *Develop an hypothesis to investigate*
- *Design and develop an experiment and collect the data (with an understanding of what the data attributes and range(s) should be)*
- *Analyze the data*
- *Interpret the data and draw conclusions that will serve as a starting point for any new hypotheses*
- *Document results*
- *Retest (if applicable)*

For less complex and lower risk SSR applications, some, or much, of the recommended information may not be available. However, while adapting a risk-guided approach, the researcher should document the software quality engineering support that is available as much as reasonably possible

A9.0 General Development (Section 9.0 of the SSR Plan)

This section should describe the general approach for implementing the requirements as understood. Prototyping and agile methods such as "Feature Driven Design" or "Test Driven Development" may be appropriate and useful for the development of software supporting research. The use of computational steering, if possible, may benefit the SSR development.

Similarly, for any software maintenance efforts, whether adaptive, corrective or perfective, the anticipated modifications, deletions, or additions should be reviewed for correctness, completeness, feasibility, verifiability, and unambiguousness before implementation.

A10.0 Software Quality Engineering Support for Implementation (Section 10.0 of the SSR Plan)

This section must specify what recommended software quality engineering practices for the software supporting the research should be documented. Reviews, checklists, and test cases are all recommended to support the software quality engineering objectives for the software supporting the research.

A11.0 Evaluation of Software (V&V) (Section 11.0 of the SSR Plan)

A11.1 Static Tools (*Section 11.1*)

If there are any software tools available to support the static analysis of the software supporting the research, the software tools and their application and limitations should be documented.

A11.2 General Testing Philosophy (*Section 11.2*)

This subsection documents the anticipated approach to testing the software and whether generally accepted white-box, black-box or coverage (C0 or C1) analysis will be used. Computational steering may also be useful to verify any intermediate code results and whether they conform to expected results.

A11.3 Criteria to Evaluate Software (*Section 11.3*)

This subsection should describe the approach to evaluate the correctness of the software supporting the research. Hand calculations or statistically comparing the results against known and accepted software results are acceptable means.

Both the successful and failed end conditions for the software supporting the research should be documented to provide criteria to evaluate whether the application achieved its desired results for supporting the research.

A12.0 Maintenance of Software Supporting Research (Section 12.0 of the SSR Plan)

So long as all the work products associated with the SSR are being maintained and managed under configuration control and configuration baselines are established and documented, maintenance activities can be easily managed. When any change is needed either as a result of new or different functional requirements, new hardware or operating system, a quick analysis will indicate what work products must be changed.

If the SSR must be modified, the same processes as noted in Sections 4.5 and 5.8 are applicable. Again using a risk-guided approach, appropriate retesting of the SSR may be beneficial to indicate the software will function as intended with the new functionality incorporated.

A13.0 Transferring the Research Results (Section 13.0 of the SSR Plan)

The researchers should be encouraged to publish the non-proprietary software results supporting their research in professional refereed journals, collaboration with peer researchers at national laboratories, industry and academics. Inclusion of lessons learned and guidance for reproducibility are also recommended.

Appendix B: Definitions

1. **Agile Development –** an iterative and incremental (evolutionary) approach to software development which is performed in a highly collaborative manner by self-organizing teams with just enough ceremony that produces high quality software in a cost effective and timely manner which meets the changing needs of its stakeholders. [definition from Scott Ambler, a signatory to the "Agile Manifesto"]
2. **Algorithm –** a finite set of well-defined rules for the solution of a problem in a finite number of steps; for example, a complete specification of a sequence of arithmetic operations for evaluating sine x to a given precision. [Ref. 5]
3. **Baseline –** (1) a specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures. (2) A document or a set of such documents formally designated and fixed at a specific time during the life cycle of a configuration item. [Ref. 5]
4. **Black box testing** (or "functional testing) **–** Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. [Ref. 5]
5. **Computational steering –** the action taken by researchers to change parameter values for the purpose of seeing immediate feedback of the impact of the change of data value. [Ref. 3]
6. **Configuration audit –** [Ref. 5]
7. **Configuration identification –** An element of configuration management, consisting of selecting the configuration items for a system and recording their functional and physical characteristics in technical documentation. [Ref 5]
8. **Configuration Management –** a discipline applying technical and administrative surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements. [Ref. 5]
9. **Coverage –** any metric of completeness with respect to a test selection criterion, i.e., the degree to which the source code of a program has been tested. This is a form of testing that inspects the code directly. "C0" Coverage is the percentage of each source line of code that has been executed. "C1" Coverage is the percentage of conditions or paths through the source code that have been executed. [Ref. 13]
10. **Error –** An error or "defect" is the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. [Ref. 5]
11. **Executable –** Also known as "machine code". Computer instructions and data definitions in a form that can be recognized by the computer processing unit. [Ref. 5]
12. **Fault –** an incorrect step, process, or data definition in a computer program. [Ref 5]
13. **Feature Driven Development (FDD) –** an iterative and incremental software development process. It is one of a number of Agile methods for developing software and forms part of the Agile Alliance. FDD blends a number of industry-recognized best practices into a cohesive whole. These practices are all driven from a client-valued functionality (feature) perspective. Its main purpose is to deliver tangible working software repeatedly in a timely manner. [Ref. 4]
14. **Integration Testing –** Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them. [Ref. 5]

15. **Life Cycle –** 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. [Ref. 5]
16. **Maintenance –** The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment. [Ref. 5]
17. **Pedigree** – Within the context of software, pedigree refers to the established and recognized history of the application and demonstrated confidence that the software application functions as intended without any unintended adverse affects. [adaptation of Merriam-Webster definition for "pedigree", Ref. 6]
18. **Protoyping** – a hardware and software development technique in which a preliminary version of part or all of the hardware or software is developed to permit user feedback, determine feasibility, or investigate timing or other issues in support of the development process. [Ref. 5]
19. **Research Plan** – a document that describes the management systems used for the planning, performing, documenting, assessing, and transferring of research results in basic and applied research institutions. (Please Note: ANSI/ASQ Z1.13-1999 uses the term "Research Plan", the terms "research proposal or experimental proposal are also acceptable terms.) [Ref. 1]
20. **Research Software** -- Research software is software that enables the research to investigate a problem domain using the scientific method to derive a solution in support of information discovery and innovation. Typically the algorithms within research software are fairly fluid as opposed to production software where little algorithm variability occurs. [this document]
21. **Risk** – the degree of uncertainty in the quantitative estimates established for resources, cost and schedule. [Ref. 12] Mathematically, risk is defined as:

    Risk = a combination of $(\text{likelihood of occurrence of a specific adverse event})_i$,
    $(\text{consequences if event did occur, whether health, programmatic or environmental. consequences})_i$,$(\text{time-frame of occurrence})_i$

22. **Scope Creep –** The continual enhancement of the requirements of a project as the system is being constructed. Scope creep occurs frequently in information systems development and is often responsible for going way over budget when the changes occur in the coding and testing stages rather than in the early design stages.[Ref. 6]
23. **Source –** the language used to write a computer program. [Ref. 5]
24. **Stakeholder** -- An organization or person with an interest or share in the research performed at the laboratory and with something to lose or gain depending on the quality of the research. Stakeholders might include a scientific community, the editors of a peer-reviewed journal., future generations who benefit from the knowledge produced, or the technological process and products that emerge from the research. Stakeholders are not customers of the research in the same way as those who provide funding. [Ref. 2]
25. **Status accounting –** An element of configuration management, consisting of the recording and reporting of information needed to manage a configuration effectively. This information includes a listing of the approved configuration identification, the status of proposed changes to the configuration, and the implementation status of approved changes. [Ref, 5]

26. **Test Driven Development (TDD)** - a software development technique consisting of short iterations where new test cases covering the desired improvement or new functionality are written first, then the production code necessary to pass the tests is implemented, and finally the software is refactored to accommodate changes. The availability of tests before actual development ensures rapid feedback after any change. Practitioners emphasize that test-driven development is a method of designing software, not merely a method of testing. [definition from Scott Ambler, a signatory to the "Agile Manifesto"]
27. **Unit Test –** Testing of individual hardware or software units or groups of related units. [Ref. 5]
28. **Use Case –** a description of the operation of the software system from a user vantage. [Ref. 17] (a Google search will also provide lots of good information about use cases and use case models.)
29. **Validation** -- The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. In terms of simulation software, validation is defined to mean "substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model. [Ref. 15]
30. **Variant –** For configuration identification purposes, a variant change denotes textual edits such as incorrect spelling or punctuation. The variant number is the integer number to the right of the decimal point. [adaptation of Merriam-Webster definition for "variant", Ref. 6]
31. **Verification** -- The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. In terms of "simulation" software, verification is used in the comparison of the conceptual model to the simulation representation to assure that the implementation is correct. [Ref. 14]
32. **Version –** For configuration identification purposes, a version change denotes a substantive change such as functional differences. The version number is the integer number to the left of the decimal point. [Ref 5]
33. **White box testing –** Testing that takes into account the internal mechanism of a system or component. Types include branch testing, path testing, statement testing. [Ref 5]
34. **Work Products** -- Any artifact produced by a process. These artifacts can include files, documents, parts of the product, services, processes, specifications and invoices. A key distinction between a work product and a product component is that a work product need not be engineered or be part of the end product. [Ref. 2]