Guideline for Professional Engineers Developing Software for Safety Critical Engineering Applications

CONTRIBUTORS

Roger Yiu Ming Cheung, P. Eng.
Jeffrey Coulson, P. Eng.
Eugen Malea, P. Eng.
Corneliu Muntean, P. Eng.
Nick Pfeiffer, P. Eng., Ph.D. (Chair)
Anton Pop, P. Eng.
Paul Spagnolo, P. Eng.
Pak Tse, P. Eng.

Notice: The Professional Standards Committee has a policy of reviewing guidelines every five years to determine if the guideline is still viable and adequate. However, practice bulletins may be issued from time to time to clarify statements made herein or to add information useful to those professional engineers engaged in this area of practice. Users of this guideline who have questions, comments or suggestions for future amendments and revisions are invited to submit these to PEO using the standard form included in the following online document:

http://peo.on.ca/index.php/ci_id/23427/la_id/1.htm

TABLE OF CONTENTS

1. PEO PURPOSE FOR GUIDELINES

Professional Engineers Ontario (PEO) produces guidelines for the purpose of educating both licensees and the public about best practices.

For more information on PEO's guideline and development process, which includes PEO's standard form for proposing revisions to guidelines, please read our document:

http://peo.on.ca/index.php/ci_id/23427/la_id/1.htm

For a complete list of PEO's guidelines please visit the Publications section of the PEO website.

2. PREFACE

Professional Standards Committee formed a subcommittee of engineers from a variety of practice areas who had experience developing software in their professional engineering practice. The group was asked to investigate the legal, ethical and technical aspects of software design and development which could have an impact on public safety and welfare. Furthermore, the subcommittee was instructed to prepare a guideline to deal with the development of software affecting public safety and welfare.

The subcommittee met for the first time on July 21, 2010, and submitted a completed draft of this document to the Professional Standards Committee for approval on January 15, 2013.

Following consultations with engineers and other stakeholders, the final draft was approved by Council at its meeting on _____, 2013.

Note: References in this guideline to engineers apply equally to professional engineers, temporary licence holders, provisional licence holders and limited licence holders. Practitioners as defined in the Professional Engineers Act, which from onwards will be simply referred to as the Act, refers to engineers and firms holding a Certificate of Authorization to offer and provide engineering services to the public.

3. PURPOSE AND SCOPE OF GUIDELINE

Software - either directly or indirectly - may affect the health, safety, security, and financial welfare of the public. The purpose of this document is to outline the ethical and professional responsibilities of engineers to ensure that the public interest is protected. This document also provides guidance for others interfacing with engineers who are developing software, such as clients and owners who are acquiring ready-made software or specifying requirements for new software. This guideline should be regarded as an addition to, but not a substitute for, specialized software training and assumes the reader is familiar with software development.

The development of some categories of software is considered to fall within the scope of professional engineering in Ontario when it is used in a manner that affects the safety or welfare of the public. In June 2008, PEO council approved the following definition of Software Engineering:

Software engineering is deemed to fall within the practice of professional engineering as defined by the Professional Engineers Act:

- *Where the software is used in a product that already falls within the practice of engineering (e.g. elevator controls, nuclear reactor controls, medical equipment such as gamma-ray cameras, etc.); and*

- *Where the use of the software poses a risk to life, health, property or the public welfare; and*

- *Where the design or analysis requires the application of engineering principles within the software (e.g. does engineering calculations), meets a requirement of engineering practice (e.g. a fail-safe system), or requires the application of the principles of engineering in its development*.

Software that meets all three criteria of the above definition is considered to be Safety Critical Software.

This guideline suggests considerations for developing Safety Critical Software and for incorporating such software as part of a larger system. The intent is to address factors that are reasonably necessary for the protection of the public including: software requirements, software design and construction, software process engineering, software quality, software assets management, and management of software projects. The practices, procedures and controls presented here must be tailored to each project's specific requirements. They are strongly recommended, but are not considered mandatory by the association.

This guideline does not deal with use of engineering software to perform calculations, modeling, and optimization analysis as part of professional engineering services or to provide information used as the basis for professional engineering decisions, judgments and opinions. The use of such software is the subject of a separate guideline Professional Engineers Using Software-Based Engineering Tools.

For the remainder of this document, software and software development activities are assumed to refer solely to Safety Critical Software unless explicitly noted otherwise.

The scope of this guideline has been necessarily limited to the professional responsibilities of professional engineers developing software for use in Ontario. Software that is developed for use out of Ontario is outside the jurisdiction of PEO; engineers should consult with those authorities having jurisdiction for their guidelines and standards of practice.

Only engineers or those supervised by an engineer can carry out the development of Safety Critical Software. Furthermore, companies who develop Safety Critical Software as part of a service to the public for use in Ontario must have a Certificate of Authorization from PEO.

Engineers often develop software that is not engineering related (i.e., business, financial, or tax software; e-commerce software; database analysis software; gaming software). Such software does not require to be sealed as part of engineers' regulatory obligations. Only Safety Critical Software, as previously defined, needs to be sealed.

4. INTRODUCTION

At the turn of the 21st century, machines are very intelligent and will become more sophisticated as we go forward in time, while the use of Safety Critical Software embedded in the machines has become ubiquitous. As we relinquish manual control over to sophisticated machines, we will enjoy the benefits of productivity improvements, but we may also witness catastrophic failures with damages in the billions, loss of human lives, environmental destruction, and creating misfortune for the world to bear. It is important to note that engineers are always professionally responsible for their work including the design and development of Safety Critical Software.

Engineers often play a key role in designing and developing Safety Critical Software. The intent of this document is to provide guidelines for engineers developing Safety Critical Software to establish effective software processes that can balance cost and quality, and to help gain public trust over time for sophisticated fail-safe machine designs. Engineers may be involved in the development of Safety Critical Software, failure of which may pose risks to public safety and security. Professional engineers must be aware of these risks and of their ethical and professional responsibilities to protect the public.

Safety Critical Software can have many applications, including but not limited to:

- Control and data acquisition,
- Sensing and interpretation software and modeling and design software that is used to make or automate critical decisions and actions that impact the public safety or security, or utility (telecommunications, water, electricity, gas, traffic) control and protection,
- Public transportation control systems and industrial safety,
- Protection and control systems software.

It is the responsibility of the sealing engineer using third party software to validate results obtained from the software used before implementing them into the system.

There is a need to provide proactive means to defend public interest, safety, and security as it may be affected by software system failure. This guideline focuses on the professional responsibilities of engineers in developing Safety Critical Software or incorporating such software as part of a larger system or product for use in Ontario. This guideline reinforces our regulatory jurisdiction while pragmatically protecting the public interest in this area.

Many other organizations have developed best practices, guidelines, and applicable standards - some of which are listed in Appendix 1. Engineers are advised to direct themselves to these other references, as required. It should be noted that this guideline is neither a standard nor a body of knowledge. It provides recommendations not prescriptive rules and does not explain theoretical or practical knowledge.

It is recognized that the engineer is most often part of a product or software development team that includes others such as Information Systems Practitioners (ISP), computer

scientists, and technologists. This document reviews the responsibilities of professional engineers to ensure that public interest is protected, by:

- recognizing professional and ethical responsibilities with software development and use, especially safety and security considerations;
- accepting professional responsibilities in product delivery (i.e., final review and sealing);
- delineating responsibilities for multi-disciplinary projects (i.e., hardware and software interfaces); and
- recognizing professional responsibilities of engineers in the different software development roles and during the various stages of software development.

The development of Safety Critical Software requires the same degree of review and validation as the development of any safety system, structure or device that can impact the public. Software is intangible and it may not be viewed with the same level of importance as drawings, hardware or systems. Nonetheless, being inherently fluid, software can be easily modified when compared with a structure or device; due to this fact Safety Critical Software requires an enhanced level of attention to functionality, documentation and version control (see 5.6 Support). This guideline identifies the responsibilities of professional engineers so Safety Critical Software development is undertaken with the level of care and diligence that is required of all engineering activities as covered by the *Act*.

All engineers are professionally responsible for the engineering work that they produce. Article 72(2)(b), O. Reg. 941 under the *Act i*dentifies one criterion of professional misconduct as “failure to make reasonable provision for the safeguarding of life, health or property of a person who may be affected by the work for which the practitioner is responsible, ”.Engineers must be aware that the concept of “reasonable provision” applies to the development of Safety Critical Software, as it falls under the practice of engineering.

5. PROFESSIONAL RESPONSIBILITY

5.1 Public Welfare

Developing Safety Critical Software is a complex undertaking comprised of specifying, designing, implementing, testing, verifying and validating. There is risk associated with interpreting the needs of clients and consumers, balancing budget and schedule constraints, and ensuring the efficiency, effectiveness, integrity, security, privacy, safety, and quality of the software. Engineers contribute to the success of software development projects. Furthermore, engineers are reminded of their responsibility for minimizing the risk of failure and protecting the public interest.

5.2 Risk Mitigation

Engineers are reminded of the importance of adopting well-recognized software engineering processes in order to mitigate risk to public safety and security (for some examples of these processes reference Appendix 1). Furthermore, engineers should be aware of relevant software system failures in the past, and be cognizant of their obligations in systems of similar nature. Refer to Appendix 2 for case studies of software system failures.

Safety Critical Software shall be sealed to provide assurance that the engineer responsible for developing the software has fulfilled their obligations under the *Act.* Furthermore, the seal provides traceability in case the engineer responsible for developing the software needs to be contacted.

5.3 Human Factors

Human errors have played a significant role in catastrophic system failures throughout the industrialized world for the past several decades. Human error prevention is of paramount importance in the design and deployment of Safety Critical Software. Human interface design, required to operate or maintain the system, should account for human capabilities and limitations in addition to meeting the necessary obligatory requirements of pertinent regulations.

5.4 Code of Ethics

Engineers are reminded of PEO's *Code of Ethics,* which states that “A Practitioner shall, regard the Practitioner's duty to public welfare as paramount.” Hence practitioners should minimize risk to public safety through use of a well-recognized software development process, with system safety considerations the foremost element in the design.

The *Code of Ethics* also states “It is the duty of a Practitioner … to act at all times with knowledge of developments in the area of professional engineering relevant to any services that are undertaken.” This provides an obligation for engineers to have knowledge of well-recognized software engineering processes as well as similar Safety Critical Software systems, including failures. Appendix 2 contains more information on malicious failures and cyber security.

Engineers must bear in mind that their duty to protect the public welfare is their highest duty, superseding confidentiality issues between clients and engineers. For more information on the duty to report please refer to the *Professional Engineering Practice* guideline.

Engineers often develop software that is not Safety Critical Software. Such software does not require sealing as part of the engineer's regulatory obligations. However, engineers must recognize that their professional and ethical responsibilities remain the same, even if sealing is not required.

5.5 Legal

As this is an area where conflict can arise, it is important to document intellectual property rights and ownership, as well as client relationships properly and get legal advice when needed. Intellectual property rights and ownership includes copyright, patents, industrial design rights, "trade secrets" and trademarks. In addition to legal obligations, the engineer also has important ethical and professional obligations. Following are excerpts from PEO's *Professional Engineering Practice Guideline,* which discusses confidential information:

- Engineers should not divulge any information sensitive to their clients' or employers' business to third parties, unless expressly or implicitly authorized by their clients or employers or required by law to do so.

- Engineers are also expected to avoid using (confidential) information for the benefit of themselves or third parties, or to their clients' or other practitioners' disadvantage. Engineers are expected to decline employment or a commission that would require disclosure of such information.

- It is generally considered that engineers may apply any general knowledge or expertise, as long as it falls into a "state of the art" category.

5.6 Support

Software risk is further mitigated when the software is adequately supported throughout its lifecycle, from requirements definition to production, maintenance and deployment. Such support may include:

- documentation such as requirements and specifications, verification and validation reports, manuals, critical function/alarm restoration/preservation processes, safety assessment reports, software version control process, software defect tracking process

- statements of conformance to applicable standards and of any limitations or restrictions

- training, where required by agreement, on installation, maintenance, and operation

- access to source code under suitable contractual terms
- expressed warranty, liability limitations (e.g. connecting to 3rd party middleware)

5.7 Multi-Practitioner Projects

Documentation related to Safety Critical Software shall be sealed by the engineer responsible for the development of the software. In cases where the software is developed by multiple engineers, for example in a large or multi-disciplinary project, each engineer may seal the portion of the documentation for which they are responsible. Such sealing shall provide assurance to the engineer responsible for the overall development of the software that the sealed portion of the Safety Critical Software has been developed in accordance with the *Act*. It is not sufficient that each portion of Safety Critical Software is sealed; the overall software should be sealed by the engineer with responsibility for the entire software.

Engineers who seal Safety Critical Software shall ensure that their obligations under the *Act* are fulfilled, including taking responsibility for portions of the software that have been developed by others but not sealed. As per PEO's guideline *Use of the Professional Engineer's Seal*, "For a project covering work within several engineering disciplines, all documents within a particular engineering discipline must be sealed by the engineer taking responsibility for work within that discipline, with an indication or qualification of which discipline is implied by the seal. The supervisory or coordinating engineer should also apply his or her seal to indicate that the work of the various disciplines has been coordinated. If only one signature and seal is used, it should be that of the engineer taking responsibility for the work, generally the coordinating engineer."

Engineers who use Safety Critical Software as part of a larger project shall ensure that the software is fit for use in the particular application, and operating and physical environment. Such fitness for use may include an understanding of functionality, reliability, usability, security, limitations, underlying principles, design constraints, and validity/reliability of results.

6. INTELLECTUAL PROPERTY

Intellectual Property (IP) refers to a variety of intangible assets such as copyrights, trademarks, patents, and trade secrets. Owners of these assets may grant legal rights restricting use. When the asset is software-related, such as the source code for a program or the details of an algorithm, the granted rights may limit the conditions under which the software can be used, maintained, or included with other software.

Modern software development is often accomplished by combining original software modules with pre-existing/re-usable software modules. The end result is a new software product with unique characteristics and functionality. In the process of software development the engineers might act as either a user of IP, or author of IP, or even both user and author within the same project.

Engineers as users of IP, when using 3rd party software or development tools should acknowledge and respect the IP rights granted or limited by licenses, warranties, redistribution statements, and disclaimers. The IP rights and their implications regarding safety, maintenance, upgrades and use by customers or other parties should be an important consideration for any engineer. In particular, a too restrictive 3rd party IP rights might limit the possibility of comprehensive evaluation of safety related features for the software module that is intended to be included into the newly developed software product (e.g.: restricted access to data constants covered by 3rd party IP limitations).

Engineers as authors or/and owners of IP should take necessary actions to protect their rights. This becomes extremely important when the IP relates to Safety Critical Software and Systems. Engineers should evaluate the potential IP generated by software development and identify technical limitations in relation to the IP. For example IP could cover technically strategic algorithms, specialized calculations, data constants obtained through extensive empirical research or the actual source code.

Engineers should engage IP legal counsel or seek legal advice any time an IP related matter is identified; in more complicated and sensitive cases the Canadian Intellectual Property Office may be contacted. Due to the complexity of IP rights, the engineer either as user or as author/owner of IP, should initiate a dialogue with trained professionals in IP laws. Some of the legal clarifications may cover, but are not limited to, the rights granted or limited by copyright, license, warranty, redistribution statements and disclaimers, in order to limit liabilities and avoid creating a risk to safety and public welfare.

More general legal aspects of engineering work in regards to safety critical systems is presented in section “5.5 Legal”

7. SEALING

Section 53 of the *Act* states: “Every holder of a licence ... shall sign, date and affix the holder’s seal to every final drawing, specification, plan, report or other document prepared or checked by the holder as part of the service before it is issued.”

The basic purpose of sealing, or authenticating, hardcopy or softcopy professional documents (refer to section 7.2 for more information on professional documents), or a software package is to identify work has been performed by or under the supervision of an engineer. The product of engineering work is sealed to indicate that other persons can rely on it to be suitable for its intended purpose. PEO requires all final safety critical software packages prepared as a service to the public, including code and documents, to be sealed in a manner acceptable to PEO.

7.1 Safety Critical Software package

7.1..1 Originally developed Safety Critical Software package

The original version or modifications of the safety critical software package, including code and documents, shall be sealed. The safety critical software package may be sealed on the equivalent of the 'cover page' or introduction to that software.

7.1..2 Review of Third Party Software

Often, software is developed for one purpose, but used for another. For example, a commercially available graphical user interface developed for a non-critical information kiosk application may be used in a critical control application. If such software becomes Safety Critical Software by its incorporation into a different purpose then the engineer responsible for the system, process, equipment or machine shall ensure that that their obligations under the *Act* are fulfilled with regards to the software. Developers of Safety Critical Software should ensure that re-use or integration with other products is properly documented, including any limitations or constraints. Where this is not the case, the engineer must review and verify that the software functions as anticipated for the intended application.

The engineer’s obligations include ensuring public welfare, providing proper credit for engineering work, and maintaining professional competence. These obligations can be fulfilled by a thorough review that includes sufficient research, calculations and other professional engineering work so that the engineer is satisfied that the work is safe and meets appropriate codes and standards. A review does not necessarily imply a complete rework. The test that should be applied is "Does the work meet the acceptable professional and regulatory standards?", not "Is this the way that I would have performed the work?". The practitioner should create and seal the documents detailing the review process.

7.2 Professional Documents

Professional documents for Safety Critical Software, either in electronic or hard copy form associated with but separate from software code shall be sealed. This may include the

following design, testing, and commissioning documents. Please note this list is not exhaustive.

- design documents, requirements, specifications; including documentation of operating environment (compiler, software versions, etc.)
- physical models of monitoring and control systems (i.e., process drawings, mechanical drawings)
- artifacts produced during the course of design and development (i.e., tradeoff analysis, prototypes, analysis elements, software defect tracking, verification and validation test reports)
- purpose and appropriate use of software including limitations and constraints and re-use or integration with other products, development and maintenance documents, tools, and aids
- review documents for third party software, as described in 7.1.2

The appearance of an engineer's seal on a professional document is taken as indicative of an engineer's involvement and acceptance of responsibility for design, development and testing. These principles apply to both paper documents and electronic documents. Refer to PEO's Guideline *Use of the Professional Engineers Seal* for more detailed information.

7.3 Revisions

When a safety critical software package undergoes a revision, then the engineer responsible for the revision shall label and seal the software code and professional documents that are part of that safety critical software package release. The seal indicates an engineer's acceptance of responsibility for the revised package and associated engineering work. Care should be taken in documenting the revisions to clearly identify the boundary of professional responsibility between the original and revised software code and documents.

8. SOFTWARE DEVELOPMENT METHODS AND PROCEDURES

There is no best single process for developing software. However, many methodologies, and standards have been created relating to software development in order to control the process and reduce the variability. Appendix 1 contains a list of well-recognized processes for software development.

Regardless of the particular method, development of safety critical software is a project that is comprised of many steps and can be managed by established techniques.

The engineer should choose a software development methodology that is appropriate for the type of software to be developed. Whatever development methodology is chosen there should be a clear documentation of why it was chosen and the benefits and risks of choosing it.

This guideline does not specify a particular development methodology. However, engineers should choose a reliable methodology, and put appropriate controls/processes in place that minimize any inherent risks.

In general, software development projects can be divided into the following major tasks:

- requirements
- design and development
- process engineering
- software quality
- assets management
- project management
- software packaging

Engineers often are significantly involved in all these different tasks.

8.1 Software Requirements

The software requirements are elicited using processes and tools for identification of the context, the end users needs, re-use or integration of existing products, code or applications, and the constraints. These requirements are then used to describe and document the functional and non functional aspects of the software. The requirements are validated through prototyping, review, and modeling. Possible failure modes need to be identified in Safety Critical Software in order to reduce probability of system failures

8.2 Software Design and Development

Software design uses methods, techniques, and tools to make and capture decisions as to how the software will be built. It covers: architecture, methods, notations, user interface, security and safety, data engineering, real-time design, and performance engineering.

Once the design has been defined, the software is developed through programming and integration of software components. This stage would also include creation of development documentation, user documentation and training materials.

8.3 Software Process Engineering

Proven methodologies in developing software include the definition, measurement, and improvement of a software engineering process in order to create a repeatable, predictable development method or life cycle. There are a number of process models which formalize the task of developing software or use project management techniques to better control the development process. Software processes can be described and measured to assess the effectiveness of the process and allow for improvement.

8.4 Software Quality

During the development, it is necessary to ensure that the software product adheres to standards, conforms to its requirements, and meets its end-user needs. Software quality can be assessed through a number of techniques including verification, validation, measurement, reviews, and audits. Testing can be used to verify proper operation and validate whether it fulfills its intended purpose. This testing can be performed at different stages of the development process including unit or component testing, integration testing between components, system testing, and system integration testing between systems. However, in general, testing can never completely identify all the defects within software. In addition, validation is subjective and contextualized based on the reviewer. Therefore, review and participation by the client (contract) and/or end-user in testing may be required for acceptance of the final product. This can include simulation, review of sample input and output data, performance testing, acceptance testing, qualification (add to definitions) and certification.

8.5 Software Assets Management

Software must be managed through its life cycle. Software asset management is used to control the software product and safeguard the versions and variants of artifacts and work products of software development. Software asset management requires identification of all software elements, their version and history, their relationships, and archival data. Change control is implemented for identification and tracking of changes to software elements, including problem reports, contract changes, constraint changes or evolution. Release management and delivery is the preparation of software for release, distribution, installation and deployment into operation. Deployment represents the activities that make the software available for use and is affected by the target physical environment, architectural and design constraints, and security and performance concerns.

8.6 Management of Software Projects

Software development needs to be effectively managed just like any other engineering project in order to measure and control the progress of the software project. Software project management uses planning and scheduling to define the types of process lifecycles, work breakdown structure, and estimations of workload. The management includes project tracking and metrics of progress, quality, expenditure, etc. In addition to the initial development, software needs to be maintained over its life cycle.

8.7 Software Packaging

Delivery of the software will require the packaging of the software or application in an electronic format that allows the user, be it a client or another project group or team to install, -understand or utilize the software. In addition to the software instruction and training materials, the applicable professional documents should be transmitted to the user. This documentation and information can be transmitted electronically or through traditional means. In either case all engineering documents and safety critical software should be sealed.

9. SUMMARY

Throughout their careers engineers may be involved in the development of Safety Critical Software. The failure of Safety Critical Software systems may pose risks to public safety and security. Engineers must be aware of these risks and of their ethical and professional responsibilities to protect the public. Appendix 2 contains some case studies of software system failures.

Many other organizations have created well-recognized methodologies relating to the development of Safety Critical Software, some of which are listed in Appendix 1. Engineers are advised to direct themselves to these other references and other well-recognized methodologies, as required.

10. **DEFINITIONS**

For the purposes of this guideline the following terms and definition apply.

Fail-Safe System

In the event of a predictable system failure, the system is returned to a safe condition that minimizes risk to life, health, property and public welfare.

Reasonable provision

The concept of “reasonable provision” can be defined as a requirement for practitioners to maintain the standards that a reasonable and prudent practitioner would maintain under the circumstances.

Safety Critical Software Documentation

Documents that express a professional opinion, judgement or direction upon which someone else may rely; both hard copies and electronic copies of design documents, professional documents, requirements, specifications; including documentation of an operating environment (compiler, software versions, etc.); testing specifications; test procedures for critical components of software interfaces; interpretation of test results; implementation procedures/guides; user guides; reports or other documents that express engineering work as contemplated in the *Act* (sections 1 and 12) or Regulations (section53), or reproductions of same.

Software Safety Assessment

A 3rd party assessment that provides an objective independent safety evaluation of the software in the overall system. The assessment ensures that the software engineers/designers have considered safety aspects in their design (e.g. Fault Tree Analysis, Operational Health and Safety Analysis, etc.)

Software Risks

These risks are defined as risks to public safety caused by software system failure.

State of the Art

Highest level of development achieved at a particular time

Trade Secret

Information including but not limited to a formula, pattern, compilation, programme, method, technique or process, or information contained or embodied in a product, device or mechanism which:

(i) is or may be used in trade or business;
(ii) is not generally known in that trade or business;
(iii) has economic value from not being generally known; and
(iv) is the subject of efforts that are reasonable under the circumstances to maintain its

secrecy.

Verification

The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase [taken from IEEE-STD-610]. Verification ensures that the product was built according to the design requirements and specifications. Verification ensures that "you built it right".

Validation

The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements [taken from IEEE-STD-610]. Validation ensures that the product actually meets the user's needs, and fulfills intended use and goals. Validation ensures that "you built the right thing”.

APPENDIX 1 – Software Engineering References of Interest to Engineers

Note that this list in no way limits the responsibility of an engineer or the scope of this guideline:

Standard	Topics Covered
Capability Maturity Model Integration (Software Engineering Institute – Carnegie Mellon)	Best Practices http://www.sei.cmu.edu/cmmi/solutions/index.cfm
Software Engineering Code of Ethics (IEEE Computer Society and ACM)	Ethics http://computer.org/computer/code-of-ethics.pdf
Guide to the Software Engineering Body of Knowledge (IEEE Computer Society)	Guide to Software Engineering http://www.swebok.org
Software Engineering Standards (IEEE)	Standards Software Engineering http://standards.ieee.org/findstds/index.html
Software Engineering Standards (ISO/IEC)	Standards Software Engineering http://www.iso.org/iso/home/standards.htm
ANSI UL 1998, the Standard for Software in Programmable Components	Standard for Software in Programmable Components http://www.ul.com/global/eng/pages/solutions/standards/
NASA Software Assurance Standards	Standards for Software Assurance http://www.hq.nasa.gov/office/codeq/software/docs.htm
International Standards for electrical, electronic and related technologies (IEC)	Standards and other publications http://www.iec.ch/standardsdev/publications/

APPENDIX 2 – CASE STUDIES OF SOFTWARE SYSTEM FAILURES

Several high-profile failures illustrate the requirement for formal software engineering. Note that these case studies in no way limit the responsibility of an engineer or the scope of this guideline:

Therac-25

Engineers should design Safety Critical software that either adapts or flags changes in hardware:

In its original configuration, the Therac medical radiation treatment machine would not function unless a protective shield was in the correct position. The machine had a flawless treatment record despite the danger posed by the large dose of ionizing radiation it was capable of producing. However, the software on the new machine was supposed to incorporate this safeguard, but instead a combination of faulty sensors on the shield, a slow response time to operator inputs and inadequate feedback to the operator led to at least six accidental overexposures, *three of which were fatal*.

North-East Power Blackout in 2003

Furthermore, engineers should ensure that Safety Critical Software is adequately monitored:

The electrical blackout affected an estimated 10 million people in Ontario and 45 million people in eight U.S. states. While this was due to a combination of factors, one of the systems that failed was a computerized system that should have raised an alarm when loads on many of one company's lines started to be exceeded. Instead, the alarm stayed silent and the operators remained unaware of the problems. On older systems, each line would have had its own control and alarm systems.

Stuxnet Virus

Engineers should design Safety Critical Software so that it can handle this threat of malicious interference:

Stuxnet is software that targets specific industrial equipment. It is believed to be the first discovered worm that reprograms industrial systems. It specifically attacks SCADA systems used to control and monitor industrial processes and is able to fake industrial process control sensor signals so an infected system does not shut down due to abnormal behavior. Stuxnet includes the capability to reprogram the PLCs and hide its changes.