engineerscanada
ingénieurscanada

Engineers Canada paper on professional practice in software engineering

Engineers Canada paper - May 2016

Notice

Disclaimer

Engineers Canada's national guidelines and Engineers Canada papers were developed by engineers in collaboration with the provincial and territorial engineering regulators. They are intended to promote consistent practices across the country. They are not regulations or rules; they seek to define or explain discrete topics related to the practice and regulation of engineering in Canada.

The national guidelines and Engineers Canada papers do not establish a legal standard of care or conduct, and they do not include or constitute legal or professional advice.

In Canada, engineering is regulated under provincial and territorial law by the engineering regulators. The recommendations contained in the national guidelines and Engineers Canada papers may be adopted by the engineering regulators in whole, in part, or not at all. The ultimate authority regarding the propriety of any specific practice or course of conduct lies with the engineering regulator in the province or territory where the engineer works, or intends to work.

About this Engineers Canada paper

This national Engineers Canada paper was prepared by the Canadian Engineering Qualifications Board (CEQB) and provides guidance to regulators in consultation with them. Readers are encouraged to consult their regulators' related engineering acts, regulations and bylaws in conjunction with this Engineers Canada paper.

About Engineers Canada

Engineers Canada is the national organization of the provincial and territorial associations that regulate the practice of engineering in Canada and license the country's 295,000 members of the engineering profession.

About the Canadian Engineering Qualifications Board

CEQB is a committee of the Engineers Canada Board and is a volunteer-based organization that provides national leadership and recommendations to regulators on the practice of engineering in Canada. CEQB develops guidelines and Engineers Canada papers for regulators and the public that enable the assessment of engineering qualifications, facilitate the mobility of engineers, and foster excellence in engineering practice and regulation.

1 Introduction

The purpose of this document is to provide information and guidance to the regulators regarding the discipline of software engineering. The document provides an introductory rationale that addresses the nature of *engineering* practice in software engineering, in comparison with common software development. It is intended to help enforcement and compliance officials to identify software engineering practice that should be regulated – where it is reasonable to expect that somebody is taking professional responsibility for the work, but should not be taken to limit the validity of software engineering work that falls outside this scope.

Note that this document does not attempt to describe the entire scope and depth of the software engineering discipline. The reader is referred to the Canadian Engineering Qualifications Board Software Engineering Syllabus [1] and the Institute of Electrical and Electronics Engineers' Guide to the Software Engineering Body of Knowledge 2] for a more thorough discussion of the scope of software engineering.

Many licensed engineering practitioners in the traditional engineering disciplines are grounded in Canadian Engineering Accreditation Board accredited programs and the scope, practices and standards for these disciplines are well defined. However software engineering practitioners seeking licensure are less likely to be graduates of an accredited program in software engineering, and come from a wide range of backgrounds in industry. There are thousands of software developers practising in industry, many of whom lack the knowledge, skills and experience to practise engineering. In some jurisdictions some individuals may be qualified for a limited license, which would grant them the right to practice within a limited scope.

In order to protect the public and to prevent unqualified software development practitioners from assuming the responsibilities or the title of licensed software engineer, regulators need an understanding of the scope of regulated practice in software engineering.

This document presents a simplified tool and guidance to help regulators and enforcement personnel recognize the practice of software engineering. This includes an application of the definition of the practice of engineering to the software field as well as indicators that an activity may involve the practice of software engineering that may only be practised by licensed software engineers (i.e., the exclusive scope of software engineering) [3]. It also includes an exploration of aspects that may be practised by others in addition to licensed software engineers. The document is not exhaustive, and is only intended to add to other information and tools currently used by the engineering regulators for the regulation of this field.

This document does not address the question of limited licence, which, in some jurisdictions, may be a mechanism to allow applicants who would not qualify for P.Eng. registration to practise Software Engineering within a limited scope.

2 Scope of software engineering practice

Software engineering is a discipline that has proven challenging for the engineering profession to recognize and, therefore, regulate. For the purposes of regulation and enforcement, the scope of software engineering is consistent with the existing definition of engineering provided in *National guideline on the practice of engineering in Canada*, which states:

The "practice of engineering" means any act of planning, designing, composing, evaluating, advising, reporting, directing or supervising, or managing any of the foregoing, that requires the application of engineering principles, and that concerns the safeguarding of life, health, property, economic interests, the public welfare or the environment.

In the case of software engineering, a piece of software (or a software intensive system) can therefore be considered an *engineering work* if **both** of the following conditions are true:

- The development [4] of the software required “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software” [5].
- There is a reasonable expectation that failure or inappropriate functioning of the system would result in harm to life, health, property, economic interests, the public welfare or the environment

Typically, when evaluating a software intensive system, the clauses are considered in the reverse order; i.e.: Is there an impact to the public interest? and then; Were software engineering principles applied? Evaluation of these questions and an explanation of software engineering principles are contained in the next section.

3 Software engineering indicators

The regulation and enforcement of software engineering practice has proven to be a difficult task, since activities such as software programming may often appear to overlap with software engineering. With reference to the above definition, there are two key aspects of an activity that indicate that it falls under the exclusive scope of software engineering:

- it concerns the public interest (life, health, property, economic interests, the public welfare or the environment); and
- it requires the application of engineering principles.

The application of these two indicators to software engineering is discussed below.

3.1 Safeguarding the public interest

The first question to ask is “Does this work/act affect the public interest in terms of life, health, property, economic interests, the public welfare or the environment?” [6]

A key indicator that a software or system falls within the exclusive scope of software engineering is whether failure or malfunctioning of the software or system could present a hazard to “life, health, property, economic interests, the public welfare or the environment.” While it could be argued that the software itself cannot present a hazard, software as part of a system certainly can, and the nature of the resulting risk depends on the nature and application of the system. Clearly there is a wide range of risk levels presented by such hazards, and this can be a factor in determining if it is software engineering. For example, software that directly controls hazardous hardware would be classed as *safety-critical* [7] and so development of such software would almost certainly be considered to be the practice of software engineering. On the other hand, the failure of game software would only present a risk to the economic interests of the development company (i.e. not the public), and so would probably not be considered as software engineering by this indicator.

Development of software that forms an essential part of, or interacts directly with, a system that is itself an engineering work, and hence concerns public interest, would also normally be considered to be the practice of software engineering. It may be the case that an engineer (from another discipline) takes responsibility for the overall system. In that case the engineer taking responsibility needs to pay attention to the guidelines on adequate supervision and practising within one’s scope just as is required for all situations where unlicensed individuals participate in engineering work under the supervision of an engineer.

The definition of safety-critical software is also generally taken to include software that is intended to mitigate or recover from the results of an accident (e.g., emergency shut-down systems), and so development of such software would also normally constitute software engineering. Also software may become safety-critical by virtue of the application of the system in which it is used. For example, software in a telecommunications network switch may be safety-critical if the switch is used to support life-support or safety systems. The failure of a telecommunications network may also materially impact the economic interests of the wider public or the economy of an area, and would be considered software engineering on that basis as well.

A public interest aspect that is unique to software is that it often is responsible for protecting the security of private data. Critical personal and corporate data can be produced by, processed by, and/or protected by software. Therefore failure of such software or systems could be detrimental to public welfare, and can be life threatening in particular circumstances (e.g.

failure to protect identities leading individuals to become victims of crime). Development of these systems can therefore be considered to be software engineering.

3.2 Engineering principles

The second question to ask is “Were engineering knowledge and skills required, and were engineering principles applied to develop this product?”

Whereas impact on public interest is a property of the ***application*** of the software (what it is used to do), the application of engineering principles is a property of the ***development process*** of software (how it was designed and implemented). In addition to the general engineering principles there are also ***software*** engineering principles which characterize work in this field. At their most general, these software engineering principles include “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.” [8] Determining if software engineering principles are required involves an assessment of the development process and of the product itself.

When looking at the software development process, we must consider what *should have been* undertaken over the life cycle of the software product. We should look for both general engineering principles and software engineering principles. The software engineering principles are what make the work “more than just programming”. They include:

- Fitness for purpose

  The software or system has a known purpose or scope and requirements are defined and consistent with that scope. There is an expectation that the software or system is fit for stated purpose.

- The use of known designs and practices

  The software or system design and the various practices used during the lifecycle are defined and selected with care, as appropriate for the risk and the other aspects of the engineering problem.

- Management of risk

  Risks of various kinds are analyzed, and managed according to the engineering problem. Risks impacting public health safety and welfare are managed as a top priority, but technical and financial risks are also considered.

- Validation and verification of work

  At each stage of the lifecycle the scope of software engineering includes validation and verification of the software or system, including its components and intermediate work products.

- Reliability and repeatability of the process used

  Specific processes are defined or selected for use during the lifecycle are appropriate for the engineering problem and level of risk.

- Assumption of responsibility, traceability and liability

  The software or system may be composed of many components and have interfaces to other devices or systems, all of which must function reliably and appropriately. The assumption of responsibility for a system (engineering work) includes the assumption of responsibility for components and associated systems

Since the actual development process that was used is not always known, we can also consider characteristics of the software. If the software exhibits certain factors, then engineering principles were likely required to produce it. Any of the following factors can require the application of software engineering principles:

- Uniqueness

  Is the product particularly novel? Products for which the software design follows familiar patterns, and can likely be implemented using well known, high level, tools, will likely not constitute software engineering. Development of products where the design is not so obvious, or requires integration of technologies in novel ways, would be more likely to constitute software engineering.

- Complexity of project

  The overall complexity of the project, which could be indicated by the number of people involved in the development effort or the size of the design artefacts (e.g., source code), or the number of connected elements (both physical and virtual), gives an indication of the need for engineering principles.

- Appearance to end user

  Applications where software forms part of a larger system (*embedded* software), such that the user of the

system is not normally aware of the software, instil a stronger expectation of correct behaviour and hence a higher standard of care is required in the development process. For example, we all expect that the accelerator and brake pedals on our cars function as expected all of the time, and probably do not often consider that there is a significant amount of software involved in these systems.

- Interdisciplinary nature

  Software development where understanding the problem domain requires knowledge of other disciplines may also be software engineering due to the broad foundations in the physical sciences that are essential for the successful development of such systems. This also applies to what is often called *engineering software*—software that incorporates some knowledge of other engineering disciplines in its internal algorithms, such as may be used in performing design calculations or simulations. Although it is clear in this case that the engineer taking responsibility for the final product (i.e. that the software is being used to design) is responsible for the outputs of the software, there is a clear application of engineering principles so development of such software can be deemed to be within the practice of engineering.

- Regulatory environment

  If the software was developed for and is operated in a highly-regulated industry (e.g. aviation, nuclear, etc.) then it is quite likely that software engineering was involved and that engineering principles were required.

The fact that a product exhibits one of these characteristics does not mean that it *must* be software engineering, but exhibiting one or more is certainly a sign that it *could* be.

Lastly, we should consider the work activities. The Institute for Electrical and Electronics Engineers' (IEEE's) Guide to the Software Engineering Body of Knowledge [9] has elaborated those activities that are part of a professional software development process. They include:

- definition of software requirements;
- software design;
- software construction;
- software testing;
- software maintenance;
- software configuration management;
- software engineering management (i.e. software project management);
- use of software engineering processes (i.e. software lifecycle, process and product assessment and measurement);
- use of software engineering tools and methods; and
- software quality management.

Section 4 provides a decision tree for an assessment process based on all of these indicators.

4 Examples

There are two important cases where software engineering often applies:

1. Software is an embedded component in a device or system which is itself an engineering work.
2. A software intensive system, where software is a principle component and may perform various processing, analytic and other functions.

In the first case, a system or device may be considered an engineering work if it controls physical mechanical systems or energy sources and poses potential risks to public health and safety. While software may form an essential component of such a system or device, the entire device is an engineering work and the responsible engineer may be trained in mechanical, electrical or another engineering discipline. In this case the responsibility for the embedded software is delegated to a software engineer who would work with other engineers on matters of interface design, integration and safety.

In the second case, a software intensive system is designed and implemented on a variety of digital computer system platforms. The software may not be implemented as a traditional engineering work – a physical device or system which itself controls mechanical systems or energy sources.

Below, examples of both embedded software and software intensive systems are discussed. Some of these examples are easily recognizable as the practice of software engineering, others are not software engineering, and some illustrate the grey areas where the level of regulation may vary.

4.1 Nuclear power

Software is used in many aspects of nuclear power production, including protection systems, control systems, data logging and design. The industry has established guidelines and standards that characterize software as being safety related, or not, and identify processes that should be followed in the development and verification of such systems. [10] Development of new software that is considered to be 'safety related', which includes protection systems and control systems, is a clear example of software engineering.

The safety aspect of such systems is clear: failure of the software could result in a catastrophic failure of the reactor, release of radiation or inadvertent shutdown causing loss of power to customers. The need for the application of engineering principles is just slightly less obvious, but is well established by a review of the processes outlined in the International Atomic Energy Agency's recommendations [11] Because of the high level of risk associated with nuclear power production, and hence strict regulation of the industry, it is essential that any software development follow a rigorous and well-documented process. In addition, this software development requires interdisciplinary knowledge related to sensors and actuators and the working principles of the power plant systems.

4.2 Biomedical devices and applications

Software is used to drive or control biomedical devices such as radiation (imaging) devices, surgical robots and non-invasive instruments such as sphygmomanometers (blood pressure measurement devices). [12] In some cases medical devices interact directly with humans and must be properly controlled. Software and the overall system must be designed not only to function in an appropriate manner, but to prevent malfunction and inappropriate or unsafe operation, even in the presence of failures.

Aside from those medical devices that interact directly with humans, other medical devices or medical information systems may perform analysis or processing of medical information. Defective software could lead to erroneous processing of data, and consequently poor decisions or actions on the part of health care providers. An example of such a risk would be a medical information system in which a set of medications is cross checked for possible conflicts. Erroneous processing on the part of such a software system could result in a false negative (in which the system indicates that there are no conflicts between one or more medications when in fact there are) or in a false positive (in which the system indicates that there are conflicts when in fact there are none).

The safety aspect of such systems is clear: failure or erroneous behaviour can result in injury to patients because of this risk, medical device design, production, operation and maintenance is subject to industry specific regulation. As in other fields, it is essential that any software development follow a rigorous and well-documented process.

4.3 Avionics

Avionics is the computer hardware and software used to control and activate modern aircraft and their systems. In terms of software engineering, the task is to write software algorithms that maintain aircraft stability, control, navigation, systems status, emergency reaction, and environmental systems. These algorithms take data from sensors, pilot inputs, ground and other aircraft transmissions, and internal data bases as well as other software routines that provide the interface to the controls and other sensors. These software routines may directly control any flight hardware, automatic functioning systems, and operator displays.

In the case where the software interfaces with physical equipment, sensors, or controls, it requires deep knowledge of the flight characteristics and other systems of the aircraft. This alone can be interpreted as engineering knowledge and practice.

The safety aspect of such systems is clear: the lives of the flight crew and passengers of the airplane depend on the systems and aerodynamic algorithms being correct and functioning in all foreseen circumstances. Because of this high level of risk, a licensed engineer should design the system(s) and take responsibility for the work.

4.4 Anti-lock braking system

Anti-lock braking systems (ABS) are designed to monitor the wheels of an automobile and control braking so that the wheels do not lock up. Most modern ABS systems are computer based and use a collection of sensors, hydraulic controls and a software control system. Under hard braking, the system will control the braking pressure to keep the wheels from locking up.

The safety aspect of this system is clear: public safety is directly involved through the lives of the passengers of the automobile and other automobiles on the road and pedestrians.

There are other reasons that this work is in the exclusive scope of engineering. It is a system invisible to the user that requires knowledge of the physical characteristics of the automobile, braking system, driving environment and safety. In this particular example, an engineer will require knowledge of the software side of the system as well as the physical hardware, sensors and controls. This is a mixed discipline project and will require software, computer, and some mechanical engineering work. Therefore, only an engineer with a broad educational formation is legally able to design and build the

system, and take responsibility for the work.

4.5 Video games

The development of video games is a commercially lucrative business. At a cursory glance, however, it is obvious that such games have very little impact on public welfare, safety, health, or the environment. This is the most important and over-riding factor, and so it is clear that video game development is not in the exclusive scope of software engineering.

This does not, however, preclude software (and other) engineering principles from being applied in the development of this software, and does not restrict software engineers from participating in this work. As with other engineering disciplines, practising outside the exclusive scope of engineering does not release an engineer from upholding the standards of practice expected of all engineers, and so a software engineer practising in this area would still be expected to uphold these same standards. To be clear, however, this does not mean that a software engineer developing a video game must use the same techniques as would be applied to developing a safety critical system, but rather the software engineer must choose techniques that are appropriate for the intended application of the system.

4.6 Stock market trading software

Most modern stock traders use automated trading software to execute programmed trades. These systems are set up to monitor market conditions and, using predetermined rules, will automatically execute trades in the stock exchange they are connected to. The rules are set up by the user and represent the user's domain specific knowledge of the market.

While a single user of this type of software is unlikely to cause any market damage with defective software, thousands of users of the same software could potentially bring down a market if their systems simultaneously performed erroneous trades. These trades would normally be performed at high speed and would be unable to be audited before they were processed. There have already been examples of market fluctuations based on erroneously programmed trades. If a market crash were to be triggered by trading software, the economic and social damage could be widespread.

The public interest aspect of these systems is clear: the economic interests of the public would be adversely affected in the event of a crash. The consequences of such a crash, resulting from an incorrectly engineered system, would be beyond the ability of the legal system to compensate. This risk alone classifies this software as work needing to be performed by a licensed engineer.

4.7 Payroll software

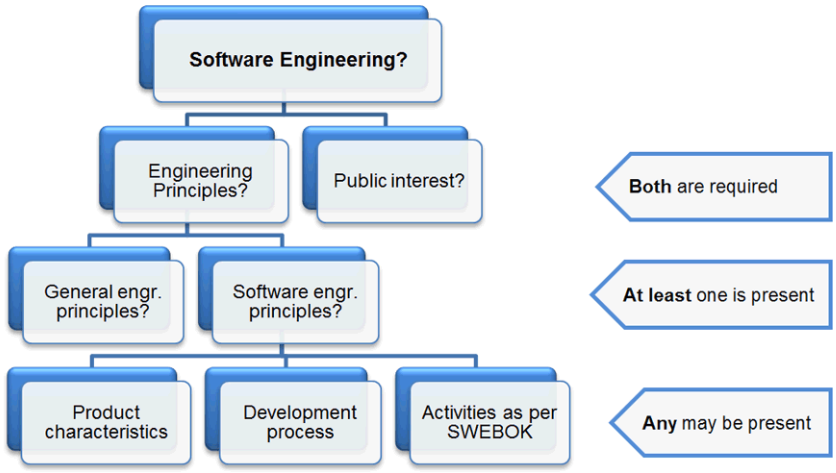
Payroll systems are used by the general public to calculate salaries and taxes by companies worldwide. These systems may print paper cheques or generate a set of Electronic Fund Transactions which are forwarded to the company's bank for execution. A commercial payroll system software product is not unique, the same software can be licensed and used by thousands of companies. An error in the calculation of the payroll could affect hundreds of thousands of individuals.

These applications systems are managed, and can be audited by, the customer/user. Before the cheques are printed or the electronic fund transfer file is sent to the bank, the responsible manager or operator of the payroll software can print payroll reports, and verify that the calculations performed by the software application are correct. This moves some of the responsibility for the correct operation of the system away from the software developer to the software operator.

This is also an example where protection against poor quality software could be provided through the legal system. It is questionable whether or not the development of software for an accounting software product constitutes the practice of software engineering.

5 Conclusion

When determining whether or not an activity or system constitutes the practice of software engineering the following questions should be asked:



The use of this framework, in combination with the characteristics and principles listed in Section 2, will allow for a more consistent understanding of the exclusive scope of software engineering practice, and will support enforcement activities in these areas.

6 Endnotes

[1] Canadian Engineering Qualifications Board, Software Engineering Syllabus, 2004. https://www.engineerscanada.ca/srv/engineerscanada-website-drupal/src/sites/default/files/syllabus_4_20.pdf

[2] Institute of Electrical and Electronics Engineers, *Guide to the Software Engineering Body of Knowledge*(SWEBOK), 2004. http://www.computer.org/portal/web/swebok/2004guide

[3] Some legislation, and the Engineers Canada definition, cited below, use the term*Professional Engineering* to refer to engineering practice that may only be carried out by, or under the direct supervision of, a licensed engineer. In this document we refer to this practice that is in the exclusive scope of software engineering as *software engineering*, whereas practice that may be carried out by non-licensed individuals is termed *software development*.

[4] In software engineering, the term “development” refers to the full product lifecycle including design, implementation, testing and sometimes installation and maintenance (fault repair and feature enhancement), as per Institute of Electrical and Electronics Engineers, Guide to the Software Engineering Body of Knowledge (SWEBOK), 2004. http://www.computer.org/portal/web/swebok/2004guide. Within this document, “development” should be understood to comprise this full range of activities.

[5] Institute of Electrical and Electronic Engineers, IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, New York, NY. http://standards.ieee.org/findstds/standard/610.12-1990.html

[6]For an illustration of the wide range of ways computer systems can be detrimental to public welfare see the Association of Computing Machinery’s Forum on Risks to the Public in Computers and Related Systems (http://catless.ncl.ac.uk/Risks/).

[7]Institute of Electrical and Electronic Engineers, IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, New York, NY. http://standards.ieee.org/findstds/standard/610.12-1990.html

[8] Institute of Electrical and Electronic Engineers, IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, New York, NY. http://standards.ieee.org/findstds/standard/610.12-1990.html

[9] Canadian Engineering Qualifications Board, Software Engineering Syllabus, 2004. https://www.engineerscanada.ca/srv/engineerscanada-website-drupal/src/sites/default/files/syllabus_4_20.pdf

[10] International Atomic Energy Agency, Verification and validation of software related to nuclear power plant instrumentation and control, Technical reports series no. 384, Vienna, 1999. https://www-pub.iaea.org/books/iaeabooks/5718/Verification-and-Validation-of-Software-Related-to-Nuclear-Power-Plant-Instrumentation-and-Control

International Atomic Energy Agency, Software for computer based systems important to safety in nuclear power plants : safety guide, Safety Standards Series No. NS-G-1.1, Vienna, 2000. https://www-pub.iaea.org/books/iaeabooks/5718/Verification-and-Validation-of-Software-Related-to-Nuclear-Power-Plant-Instrumentation-and-Control

[11] *Ibid.*

[12] Biomedical Engineering Desk Reference, Buddy D. Ratner et al, Academic Press, 2009, ISBN: 9780123746467

Heath Canada, Notice - Software Regulated as a Class I or Class II Medical Device, December 3, 2010. http://www.hc-sc.gc.ca/dhp-mps/md-im/activit/announce-annonce/md_notice_software_im_avis_logicels-eng.php

Food and Drug Administration, Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices, May 11, 2005.
http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm089543.htm