engineerscanada
ingénieurscanada

Engineers Canada paper on professional practice in software engineering

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.

About Equity, Diversity, and Inclusion
By its nature, engineering is a collaborative profession. Engineers collaborate with individuals from diverse backgrounds to fulfil their duties, tasks, and professional responsibilities. Although we collectively hold the responsibility of culture change, engineers are not expected to tackle these issues independently. Engineers can, and are encouraged to, seek out the expertise of Equity, Diversity, and Inclusion (EDI) professionals, as well as individuals who have expertise in culture change and justice.

1. Introduction

Revised April 21, 2023

In Canada, the profession of engineering is self-regulated by provincial/territorial engineering regulators, pursuant to a statutory mandate set out in engineering legislation. The delegation of regulatory function recognizes the specialized knowledge of the profession and its ability to develop and maintain standards of competency and conduct to ensure that the public interest is served and protected. In fulfilling their statutory mandates, engineering regulators are entrusted with a variety of responsibilities, including regulating the practice of engineering and the use of the engineer title.

The practice of engineering*:* According to Engineers Canada's "Public guideline on the practice of engineering in Canada", only an engineering licence holder can engage in the independent practice of, or take responsibility for, engineering work, which is defined as 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. This definition serves as a basis for this paper, but it should be noted that the definition of engineering, as well as its enforcement, may vary by regulatory jurisdiction. Each regulator has the authority to apply and interpret its own definition of engineering, as set out in its enabling legislation, as well as regulations, by-laws, or policies made thereunder.

The use of engineer title*:* With a few legislated exceptions, only engineers can represent themselves as engineers. Only engineering licence holders can affix an engineering seal to their work, which demonstrates to the public that an engineer provides the work to the high standards of the engineering profession.

Although each jurisdiction typically establishes its own code of ethics, engineering licence holders are typically bound by their code of ethics to:

- only practice in their areas of competence;
- maintain their knowledge, skills, and abilities throughout their careers; and
- hold paramount the safety, health, and welfare of the public, and the protection of the environment.

To ensure that only competent individuals practice or take responsibility for engineering work, engineering regulators set standards of practice, ethics, and continuing competence. They investigate complaints of unprofessional conduct, and impose disciplinary sanctions including suspension and revocation of an engineering license when appropriate. Engineering regulators also take action against persons who represent themselves as engineers but do not possess an engineering licence or who are actually practising engineering without a licence. The work of engineering regulators protects public interest.

If you wish to learn more about the legal requirements applicable in each jurisdiction, you can consult the individual provincial and territorial engineering regulators' enabling legislation, as well as regulations, by-laws, and policies made under this legislation.

2. Background

Growing public concerns with automated technology, the increased frequency of malicious cybersecurity events, and the rapid pace with which software is becoming integrated into all aspects of daily life are drawing increased attention to requirements to hire engineers to protect the public in these areas. This paper defines key elements of the practice of software engineering and explains the legal requirement, in most Canadian jurisdictions, for this work to be undertaken by engineers. The intended outcome of this paper is to help regulators, practitioners, and the public to be better able to differentiate software engineering from other non-engineering software work. This guidance is being provided to help regulators and the public identify when enforcement activities or reporting to engineering regulators may be appropriates against the misuse of the engineering title and unlicensed practice of software engineering. The specific legal and regulatory environment in a jurisdiction will determine how and where this guidance should apply.

This paper also provides examples of software engineering in four practice areas, discussed in linked appendices to this document, as set out below:

- **APPENDIX A** – Construction: Structural and foundational analysis and design
- **APPENDIX B** – Manufacturing: Managing workplace risks through process safety
- **APPENDIX C** – Health care: Diagnostic imaging and medical image sharing
- **APPENDIX D** – Transportation: Public transit management systems

These examples are not intended to encompass the entire scope and depth of software engineering, but rather provide an interpretation of how software engineering applies in sample areas of practice.

3. Use of the Title "Engineer"

Irrespective of practices in other countries, representing oneself as an engineer in Canada is a protected act and requires licensure under the provincial and territorial engineering Acts.[1] On July 19, 2022, all provincial and territorial engineering regulators in Canada, along with the CEO of Engineers Canada, signed a letter noting that "[u]se of 'software engineer', 'computer engineer' and related titles that prefix 'engineer' with IT-related disciplines and practices, is prohibited in all provinces and territories in Canada, unless the individual is licensed as an engineer by the applicable Provincial or Territorial engineering regulator." The full text of the letter is presented in Appendix F.

Unlicensed individuals cannot use the title software engineer in their job titles, resumes, reports, letterhead, written and electronic correspondence, websites, social media, or anywhere else that may come to the attention of the public. Employers and individuals are not permitted to refer their employees or to themselves as a "Software Engineer" (or variations that the substitute the word "software" with a related discipline, for example: "Firmware Engineer," "Data Engineer," "Network Engineer", "Process Control Engineer", "DevOps engineer", and so forth) unless the individual is licensed as an engineer.

4. Practice of Software Engineering

According to the national definition in the Public guideline on the practice of engineering in Canada, engineers "plan, design, compose, evaluate, advise, report, direct or supervise, or manage any of the foregoing, work that requires the application of engineering principles and that concerns the safeguarding of life, health, property, economic interests, the public welfare or the environment." As noted above, this definition forms the basis of the position taken in this paper. However, it should be noted that the definition of engineering, as well as its enforcement, may vary by regulatory jurisdiction. Each regulator has the

authority to apply and interpret its own definition of engineering, as set out in its enabling legislation, as well as regulations, by-laws, or policies made thereunder.

Software engineers are engineers whose area of practice is software or software-intensive systems, and so they would typically be involved in the specification, design, implementation, analysis, and validation of software. This includes the creation of software-based tools used to do other kinds engineering work. Software engineers' role in protecting the public is especially important in the case of software that is critical to the protection of life, health, or the environment, or in cases where software is substantially integrated into engineering works. A more detailed description of software engineers' work follows.

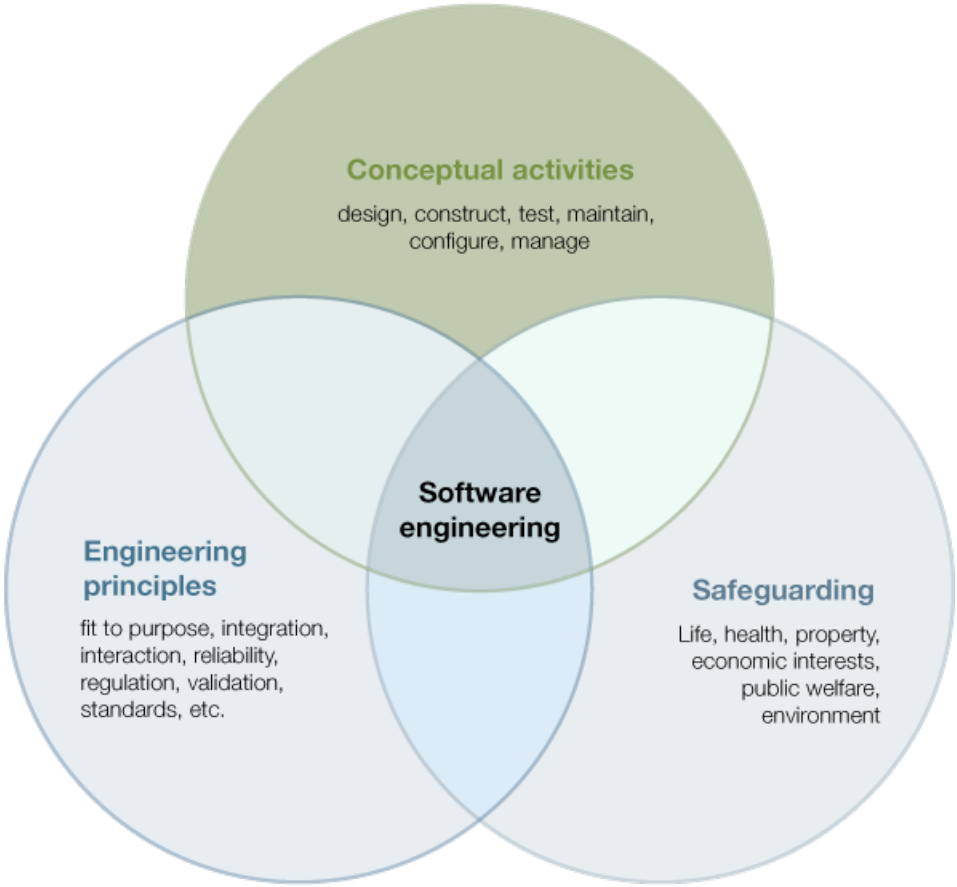

Figure 1. Essential components of software engineering.

4.1 Software engineers use particular conceptual activities or combinations of them to do their engineering work

Software engineers design, construct, test, maintain, configure, and manage software, and require understanding of the work's entire life cycle. In most jurisdictions, non-engineers, such as software developers, can perform these conceptual activities, providing that the work does not require the application of engineering principles, nor require the safeguarding of life, health, property, economic interests, the public welfare, or the environment.

However, as a general rule in most Canadian jurisdictions, when the work requires engineering principles and safeguarding activities, non-engineers may only perform these conceptual activities as part of a team where an engineer whose area of practice is within the scope of the work takes responsibility for the overall system, often referred to as direct supervision. For more general information on direct supervision, you can refer to Engineers Canada's Public Guideline on Direct Supervision. Regulators also often have jurisdiction specific requirements and expectations for engineers that are supervising engineering work performed by non-engineers.

As an exception to this general rule, in Quebec, software engineering can be carried out by non-engineers, provided they do not perform any activity reserved to engineers or represent themselves as engineers.[2]

4.2 Software engineers apply engineering principles

Throughout the lifecycle of software, software engineers:

- Consider constraints, designed to best fit the purpose or service intended and address interdisciplinary impacts of the software on public welfare;
- Analyze technical and technology risks to the public and offer solutions to mitigate the risks;
- Apply engineering knowledge to design, implement, integrate, validate, deploy, and maintain software; and
- Understand solution techniques and independently verify the results of the software.

Determining if software engineering principles must be applied involves an assessment of the development process, the desired functions of the product, and the nature of the entities and environment with which it interacts.

The presence of some or all of the following indicate that engineering principles have been applied:

- Essential functions of the software cannot be fulsomely understood without thorough analysis of the behaviours of the software while interacting with:
  - Other software items;
  - Users;
  - Physical or environmental elements; or
  - Elements already existing within the system (e.g. through regression testing)
- Essential functions of the software require interdisciplinary knowledge for core functions, to integrate with other software, or to integrate or interact with other systems or components
- The software requires integration with, and controls, regulates, or facilitates operation of, engineered work;
- The software requires understanding of the complexity around stringent requirements such as reliability, security, user interface, maintainability, testability, portability, interoperability;
- Availability, reliability, and/or maintainability are essential aspects of the software's fitness for its intended application;
- Applicable regulatory or customer requirements mandate validation and verification of work at critical stages;
- Applicable regulatory or customer requirements dictate a higher standard of care and expectations of a structured and rigorous development process, often including adherence to published development process standards; and
- Assessment of current and anticipated future software-related risks to the public or stakeholders could be substantially mitigated through the use of software engineering principles.

4.3 Safeguarding life, health, property, economic interests, the public welfare, or the environment

Engineers do not just provide technical solutions; they have an individual, professional responsibility for the impact of their designs, for the integration with other physical and virtual work with other disciplines, and for the completeness of the product. They are obligated to make ethical choices for the benefit of the public interest, even when in conflict with their individual interest.

If the engineering work requires the safeguarding of life, health, property, economic interests, the public welfare, or the environment, then, under the definition of "engineering" applied in this paper, a software engineer must take responsibility for the work.[3] Software that forms a significant or controlling part of essential public infrastructure (such as power distribution systems or essential public communications networks) or that controls systems that could put the public at risk (such as self-driving cars) clearly fits this definition, and so requires, in most jurisdictions, that a software engineer take responsibility for the system. In circumstances where the system is subject to demand-side legislation (which is legislation or regulations that require the certification of projects and works by an engineer) and requires safeguarding the public (for example in a power transmission system), only an engineer competent in software engineering can accept responsibility for the system's software components.

Indicators that software work involves the safeguarding of life, health, property, economic interests, the public welfare, or the environment include:

- **Deficient work could cause harm**: Deficient or missing function(s) of the software is likely to lead to property damage, harm to persons, or harm to the environment; and

- **Deficient work could cause catastrophic damage**: Deficient or missing function(s) of the software, however unlikely, may lead to severe harm or death of persons, widespread economic consequences, or catastrophic damage to the environment.

The risks to safety and public welfare are determined by the reasonably anticipated uses of the system in which the software is used. The requirement for sufficient safeguarding is implied by the nature, extent, and exposure of those risks with respect to stakeholders, including the general public, users of the system, and clients. In cases where safeguarding issues are potentially present, numerous factors must be taken into consideration by the software engineer:

- the current intended purpose of the work as well as reasonably foreseeable future applications over the life cycle of the software product
- all aspects of the development, deployment, and configuration processes for software associated with safety critical systems, the failure of which may cause or fail to prevent harm
- data protection, security, and infrastructure network operation as required to ensure the proper operation of the software
- the protection of public economic interests, beyond simply the interests of individual corporations or clients
- acknowledgement that the applications of computerized systems are constantly evolving, both in nature and degree, which leads to evolution of the public safety implications of such systems; and
- understanding the provenance of, assessing the appropriateness of, and defining a proper integration approach for third-party software libraries or components not designed or implemented by the software engineer.

Impact on public welfare goes beyond clients; it includes individuals and groups that might not have paid for services or products but would still suffer consequences should the software fail or is significantly deficient.

5. Employer Obligations

Every firm engaged in the practice of engineering in Canada must register with the engineering regulator in the provincial and territorial jurisdiction(s) it practices (exceptions to this requirement exist in Québec). As discussed above in Section 3, representing oneself as an engineer in Canada (including in a job title) requires licensure with the provincial or territorial engineering legislation. Employers who have employees practising reserved software engineering activities must ensure that the employees are licensed with the applicable provincial or territorial engineering regulator, or that, alternately, an engineer directly supervises and takes responsibility for the unlicensed employee’s engineering work.

6. Benefits of Hiring an Engineer

As is evident from Section 4, not all software practice is software engineering and so not everybody who develops software needs to be an engineer. Hiring an engineer gives several benefits for employers and the public, however. The engineering licensing system ensures that those who are licensed have been appropriately trained to undertake the work and have gained sufficient experience with appropriate mentorship to be qualified to take on the work independently. Engineers are also legally bound to adhere to their code of ethics and the relevant acts and bylaws of the jurisdiction in which they practice, which means, for example, that they must avoid any conflict of interest that may arise and must not undertake work for which they are not qualified. Above all, engineers’ obligation to hold the public interest paramount is the most important benefit to hiring an engineer.

There is also the benefit of accountability and oversight, as engineers are subject to the oversight by the regulators. Concerns about the practice of engineers can be brought to the regulator, which has legislated authority to investigate these concerns and take action to protect the public, where the investigation reveals that these concerns are warranted.

7. Conclusion

The software field is broad and evolving, so it would be impossible to list all types of work that constitutes software engineering. The appendices following the main body of this paper provide samples of how software engineering, as defined by this paper, applies in different specialized areas of software practice. This paper’s definition is intended to benefit regulators, who can choose to use this information to inform their enforcement practices; individuals and employers, who may draw on it to inform their practises and identify software work that may require an engineer; and the general public, who may draw on it to properly seek out licensed engineers in instances where it may be required by law.

Appendices

A note on the appendices

From safety-critical functions in industrial manufacturing or construction to integrated diagnostic tools in health care, or to the optimization in transportation, software engineering plays a variety of key roles involving analysis, specification, design, development, certification, maintenance and testing of software systems. It may occur in both established engineering disciplines and in circumstances that are not traditionally engineering (e.g. finance, communications, information management). Regardless of the context in which software engineering occurs, software engineers are bound by their jurisdictions' codes of ethics.

The sample areas of practice in the following appendices are not intended to describe the entire scope and depth of software engineering within industries, but rather to serve as illustrations of ways that software engineering, as defined by this paper, applies in a variety of contexts.

If you wish to learn more about the legal requirements applicable on each jurisdiction, you can consult the individual provincial and territorial engineering regulators' enabling legislation, as well as regulations, by-laws, and policies made under this legislation.

- **APPENDIX A** – Construction: Structural and foundational analysis and design.
- **APPENDIX B** – Manufacturing: Managing workplace risks through process safety
- **APPENDIX C** – Health care: Diagnostic imaging and medical image sharing
- **APPENDIX D** – Transportation: Public transit management systems
- **APPENDIX E** – Letter from Canadian engineering regulators on "Use of 'Software Engineer' and Related Titles in Canada"

As noted above, the position taken in this paper is based on the national definition in the Public guideline on the practice of engineering in Canada, engineers "plan, design, compose, evaluate, advise, report, direct or supervise, or manage any of the foregoing, work that requires the application of engineering principles and that concerns the safeguarding of life, health, property, economic interests, the public welfare or the environment." However, the definition of engineering, as well as its enforcement, may vary by regulatory jurisdiction. Each regulator has the authority to apply and interpret its own definition of engineering, as set out in its enabling legislation, as well as regulations, by-laws, or policies made thereunder.

The design, maintenance, and operations of systems that do not require the three elements of software engineering, as outlined in this paper, may fall outside of the scope of regulation (depending on the definition of engineering that is applied in that jurisdiction in question). The bulk of an engineer's software activities may consist of non-engineering work. Non-engineering software work may include, but is not limited to, development of non-safety-critical software functions (e.g. saving and printing functions in a software suite) or front-end application design (e.g. UX design, visual elements).

Appendix A – Construction: Structural and foundational analysis and design

Practice Area A: Construction

Focus: Structural and foundational analysis and design

Background

As evidenced by a number of structural failures recorded in the news and historical records, construction projects can be risky and complicated ventures. Today, projects often involve curved structural elements, a wide array of loads and deflections to consider, as well as the seamless integration of existing infrastructure with new structures. Among other factors, this complex work requires striking a balance among efficient design processes, structural performance, and the initial architectural vision. In this environment, structural and geotechnical engineers are constantly being pushed to become more efficient in the way they work, particularly by increasing the speed, quality, and accuracy of their designs. One method for achieving this is to develop software to perform the more repetitive, labour-intensive, and error-prone tasks that were traditionally completed manually.

Safeguarding

Construction engineering software allows the creation of models and simulations that factor in significantly more information than manual designs, enabling increased complexity and reliability in the engineering team's designs. To effectively develop such software, whether for a custom application or off-the-shelf product, a designer must be able to solicit a robust understanding of client needs, and must have sufficient knowledge to be able to test and validate the software's results. Additionally, the software must be developed in a way that anticipates reasonably foreseeable risks associated with its future uses. While ultimately the construction engineering firm assumes responsibility for design and construction of a project itself, the risks associated with the ineffective design of construction engineering software still involve the ***safeguarding of life, health, and property.*** As a result, the design and analysis software should be designed by a software engineer.

Conceptual Activities and Engineering Principles

Applications used to support construction engineering projects must be designed in such a way that they safely and reliably meet client requirements. To do this effectively requires knowledge of **engineering principles**, including the ability to consider constraints and design in a way that is fit to purpose. Notably, an application itself may only minimally require technical engineering expertise (e.g. mathematical calculation), but only someone with knowledge of engineering principles will be able to safely gather requirements and validate whether they have been properly satisfied.

Within this context, safe and reliable software design requires **conceptual activities**, which entail systematic design, construction, testing, maintenance, configuration, and management of the software. Moreover, effective development requires understanding of the work's entire life cycle. When there is a risk that a lack of proper conceptualization may cause public harm, the work should be undertaken by a software engineer.

Example: Computational mechanics

The use of computational mechanics in project design—the intersection of mechanics, applied mathematics, and computer science—has generated considerable efficiencies for the construction industry. To safeguard the public, systems employing computational mechanics should be designed or overseen by software engineers, because the proper development of such systems requires conceptual activities (e.g. systematic development and validation) and the application of engineering principles (e.g. thorough analysis of the required software capabilities and functions).

For example, computed parametric modelling can be used to determine appropriate structural configurations and geotechnical properties to enhance the seismic performance of a bridge. For this purpose, software models are created, for example, to analyze a set of ground motions with various intensities, including dynamic soil-bridge interaction effects. In the analyses, the effect of various structural and geotechnical properties (e.g., foundation soil stiffness, backfill compaction level, pile size and orientation, abutment height, and thickness) are considered. Such modelling can be both more computationally efficient and, often, more reliable than manual modelling methods. However, its safe development may require an engineer's perspective, because an engineer will have the broader knowledge to validate its outputs.

Example: Building information modelling

Building information modelling (BIM) has become a valuable tool for managing data and generating designs

during the entire lifecycle of a project. BIM applications allow for the integration of multi-disciplinary data, which can be used to model simulations that provide engineering teams with critical information to assist in their decision making and planning (e.g. pre-fabrication potential, waste management and mitigation, required materials, design conflicts, etc.). Such software has the potential to help teams identify and mitigate design and/or safety issues prior to actual construction. While a software engineer may not be required to develop all aspects of BIM applications (e.g. the application's interface), and while accountability for construction decisions ultimately resides with the construction firm, engineering expertise is required to oversee and validate any functionality that has the potential to impact safeguarding of the public and its welfare. This is particularly the case where the construction firm may not have knowledge of the application's back-end functionality.

Conclusion

Much of construction software design and development (e.g. the software's interface, data entry support, saving and printing functionality, etc.) may not require the expertise of an engineer. It is only in cases where an element of the application has the potential to impact safeguarding of public welfare, and where such risks can be mitigated with an expert knowledge of engineering principles and conceptual activities, that a software engineer should undertake or validate/oversee the development of software.

Appendix B – Manufacturing: Managing workplace risks through process safety

Practice Area B: Manufacturing

Focus: Managing workplace risks through safe software engineering

Background

Today's manufacturers face operational challenges surrounding the safety of personnel, equipment, and the environment. They must meet standards set out by industry bodies, regulation, and legislation. Process safety allows for the management of these factors through the implementation of controls to reduce risks such as high-impact fires, explosions, accidental chemical releases, structural collapses, equipment malfunctions, corrosion, component failures, and disruptions. When these risks are mismanaged, the consequences can be catastrophic.

To address risks and improve performance in manufacturing systems process control systems and safety systems (both outlined below) are often merged into a one platform, called integrated control and safety systems (ICSS). These systems, which use both hardware and software to increase automation of industrial functions, provide integrated tools that can help reduce the need for labour, increase reliability, and decrease the possibility of systematic errors. The nature of these systems means that the software used to manage and automate these systems must be designed with an understanding of how different physical and computing components will interact, and must be designed with extensive, effective cybersecurity measures.

Conceptual Activities and Engineering Principles

When designing and implementing software systems, software engineers working in manufacturing ***apply engineering principles*** by drawing on their knowledge of and integrating existing, new, and emerging technologies. The software systems they design must reliably process real-time information and must automatically manage control systems relating to manufacturing processes. This requires the ability to test and analyze the way integrated systems will interact, and a knowledge of the larger systems' entire lifecycle, including the impacts of its component technologies' obsolescence and/or the potential impacts of incorporating new components with the system in the future.

In terms of ***conceptual activities***, software engineers working in manufacturing use their understanding and operation of the development stages—analysis, design, development, testing, and maintenance. The software engineer engages in these conceptual activities to ensure, for example:

- Responsiveness of software systems to the facility's process automation systems;
- Effective quality management and quality assurance;
- Reduced energy costs and overall enhanced performance of systems within safe parameters;
- Accurate real-time information transfer and satisfaction of data visibility requirements to ensure safe monitoring;
- Peak asset performance by creating software systems that can monitor connected smart instrumentation and field devices, and manage calibration schedules and equipment maintenance;
- Software systems that allow for successful migration to new human-machine interfaces; and
- The capability of a software system to be reconfigured or adjusted for future changes to safety guidelines or other required parameter changes.

Safeguarding

Numerous dimensions of software work in manufacturing involve ***safeguarding life, health, property, economic interests, the public welfare, and the environment***. Software engineers have the knowledge needed to design software that can integrate and manage disparate manufacturing systems, based on specific criteria, for safe and effective operation. This safeguarding includes cybersecurity protection to ensure, for instance, that unauthorized configuration changes to a safety system cannot be made, and to prevent interference with the system's ability to accurately represent the status of the system's instrumentation (e.g., the loss of alarms, total loss of visibility, spoofing the operator). Software engineers also ensure that software systems safely separate control and safety components, so that an intrusion into the basic control system cannot circumvent safety instrumentation systems. In a properly engineered software control system, in other words, the failure of a non-safety-related function should not be able to cause a failure of a safety-related function.

Because manufacturing processes are increasingly managed through software systems, and because these systems require expert knowledge of the way that systems must be designed and integrated with others for their safe functioning, a software engineer should be responsible for their design.

Example: Industrial control systems

Expertise in software engineering is essential in the development and implementation of software used to manage and automate the *industrial control systems* used extensively in industries such as chemical processing, pulp and paper manufacturing, power generation, oil and gas processing, and telecommunications. Software used to manage industrial control systems must automatically receive and integrate data from remote sensors measuring process variables (e.g., pressure, temperature, level, flow), compare the collected data with desired setpoints, and derive command functions that are used to control a process through the final control elements, such as control valves. This software must also be able to safely and reliably communicate with the components it monitors and manages, ranging from a few modular panel-mounted controllers to large interconnected and interactive distributed control systems (DCSs) with many thousands of field connections. To effectively manage, process, and adjust controls based on the data coming from such varied inputs, the software's design requires understanding of the interaction of the software with physical systems, the interaction of the software with hardware and other embedded systems, and the lifecycle, limitations, and risks of such systems.

Due to the numerous, complex elements that must be seamlessly integrated into these software systems, and the safeguarding impacts that their failure could entail, they should be designed by a software engineer.

Example: Industrial safety systems

Industrial safety systems are designed to protect people, facilities, and the environment by putting in place measures that can automatically respond when processes go beyond allowed control margins. They are crucial in all potentially hazardous plants, such as oil and gas and nuclear plants. When such a system relies on a software component, such as when the processing of data points from within a gas plant is automated to trigger a safety measure when the data exceeds certain temperature parameters, the software should be designed by a software engineer. This is because a software engineer will have knowledge of the conceptual activities required for safe design of such systems, including knowledge of how to determine methods for testing and maintaining the systems so as to ensure their safe operation. A software engineer also will understand the interaction of the software system with other physical and hardware systems, and will be able to take measures to secure the system against cyberattacks (in this case, it may just be a matter of identifying the risk and ensuring a cybersecurity expert is consulted in the plant's network design).

Conclusion

Like any engineer, a software engineer is duty-bound to develop products and systems that "Hold paramount the safety, health and welfare of the public and the protection of the environment and promote health and safety within the workplace."[4] In the case of manufacturing, software engineers should define appropriate requirements, define operating use cases, and develop reliable software systems, drawing on their broader knowledge of legislation, physical systems, engineered hardware, and software security. Due to the centrality of these systems in manufacturing safety, and the catastrophic consequences of error, only an engineer competent in software engineering should be tasked with their design and implementation.

Appendix C – Health care: Diagnostic imaging and medical image sharing

Practice Area C: Health Care

Focus: Diagnostic imaging and medical image sharing

In Canada's complex and fragmented health care system, there has been widespread recognition of the role that IT and software-based solutions can play in delivering high-quality patient-centred care. Health care professionals are continuously seeking out ways to achieve better outcomes with minimal resources, needing, for example, to share data instantaneously across multi-disciplinary teams in various locations. Within this ecosystem, medical technologies and systems often require complex software integrations that both comply with legal and regulatory frameworks and are designed with safeguarding of the public at the forefront.

Conceptual Activities and Engineering Principles

Software engineers develop and oversee the development of software systems that assist in medical image processing, secure medical recordkeeping and sharing, patient diagnosis, patient monitoring, and clinical decision-making. A software engineer considers the entire lifecycle of the component, independently verifying its results and analyzing risks to its interaction with other systems, including physical systems, medical hardware, and other software.

In the field of health care, a software engineer may undertake a variety of engineering activities relating to ***conceptual activities*** and the ***application of engineering principles***, including:

- developing a software component that processes imaging information in a manner that accounts for its entire lifecycle (i.e., through design, construction, testing, maintenance, configuration, scalability, availability, and management);
- designing software components of monitoring equipment in such a way that they operate within constraints pertaining to accuracy, latency, and execution time of algorithms;
- ensuring compliance of software for medical information management in such a way that it meets requirements established within federal and/or provincial and territorial information protection and privacy laws;
- designing and development of imaging and information sharing software in such a way that it minimizes risks relating to data breaches and cyberattacks;
- creating software interfaces for medical equipment in such a way that they accurately and reliably represent medical data, developing safeguards to ensure the integrity of information in various use cases (e.g., power shortages, network failures, data corruption, cyberattacks);
- ensuring secure transmission of data in automated medical record keeping systems;
- providing quality assurance pertaining to software, user interfaces, systems interoperability, process automation, etc.;
- designing the software so it may eventually be extended and further refined, in accordance with sustainable development. This also means carrying out a technological watch so the software remains secured throughout it's lifecycle (e.g., new cyberattacks, dependant software vulnerabilities);
- recommending deployment strategies that take security, scalability, availability and resources consumption into account; and
- recommending technologies, deployment strategies, updates, fixes.

In medical fields, software engineers are often required for work in areas where broad interdisciplinary engineering expertise is required (e.g., robotics, mechatronics, virtual and augmented reality, mobile, wearable and implantable devices, health informatics). Their interdisciplinary knowledge and expertise on the interaction of systems is required to ensure safe and effective integration of software components with other systems.

Safeguarding

Doctors and other health professionals hold the ultimate responsibility for health care decisions, but accurate and reliable recording, representation, storage, and transmission of patient information is critical to medical professionals' ability to properly conduct their work. Improperly designed and implemented software in medical care environments can have serious repercussions, including incomplete or biased information, increased risk of misdiagnoses, and even loss of patient life[5]. Even in cases of medical data breaches, which pose minimal immediate risk to patient's life, medical information is considered highly personal and sensitive. Privacy breaches of medical information can be very upsetting or embarrassing for patients, and can have other impacts as well, such as identity theft or loss of insurance coverage. Accordingly, medical information is subject to extensive legal privacy protections.

Because of these risks, and because of the extensive knowledge required to properly mitigate them, software

engineers must develop or oversee the development of software components involved with medical equipment and medical information management systems. Engineers are bound by their code of ethics to ***safeguard life, health, and public welfare***, and so working with engineers over the lifecycle of software components for these systems helps medical professionals ensure that they are meeting their own professional responsibilities and requirements.

Example: Rapid diagnosis for a stroke patient

For a patient who has experienced a cerebrovascular accident (CVA), or stroke, fast, accurate treatment is critical. In the diagnostic phase, medical staff will order imaging tests to rule out other conditions and ultimately determine which type of stroke a patient has had. Immediate treatment may minimize the long-term effects of a stroke and even prevent death. Certain types of strokes, for example, can be treated with a dissolving agent if it is administered within approximately three hours of the event.

Using a conventional process, images of the patient would be manually transferred to the imaging system, where they could be searched, selected, and uploaded for analysis. The clinician would analyze and save the images, prepare the results, and subsequently transfer them to the physician for interpretation. In a system where minutes can mean the difference between life and death, automated solutions for the transmission of information offer a valuable prospect for improving patient outcomes.

A software engineer is trained to implement solutions that consider risks such as improper data management and transmission. If, for example, data status is not properly monitored and communicated to the software user, a doctor may be making decisions based on inaccurate information. Likewise, if patient information is stored or transmitted using improper techniques, it could be susceptible to data corruption and/or malicious actions. Engineers are duty bound to have an understanding of the software's function within its deployed context, and an understanding of its vulnerabilities in relation to other systems, including physical systems. As a result, engineers are uniquely positioned to mitigate against a wide range of risks that a different type of software developer would not necessarily perceive. Given these factors and numerous others relating to safeguarding the public, software engineers should be responsible for the design of safe, properly implemented medical information software systems.

Example: Automated workflows

The previous example illustrated the life-saving benefits of software-based tools to improve the speed with which information can be shared. Another key area where software has been beneficial to improving efficiency in the medical profession is in the automated analysis and rendering of data to augment decision making.

With the use of artificial intelligence-assisted automated workflows, information is integrated and prioritized automatically. In the example of patient imaging, when the patient undergoes the imaging procedure, a "zero-click" automated platform could be used to perform the following functions:

- Process and categorize images through innovative imaging modalities;
- Analyze and detect abnormality through the integration of advanced visualization algorithms, helping detect, classify, and characterize conditions at the point of image acquisition;
- Prioritize and issue alerts on results, which are based on integrated data that can be shared across all networks (regardless of imaging vendor or system).

Such an automated workflow enables efficient sharing of information allowing for the faster transmission of results for interpretation by the health professional. When properly designed and implemented, such a system has the potential to increase overall productivity and responsiveness of medical facilities, reduce manual burden and human error, and ultimately improve the patient's prospects of a favourable outcome and the overall quality of care received by patients. Collaboration between software engineers and healthcare professionals is of the utmost importance at that stage, so that system generated results may be compared to real-life results. Biases in the software's analysis may be discovered and corrected using that approach.

In the design of such a system, the application of engineering principles are required in several ways. The designer must consider hardware and physical constraints and design the automation software in such a manner that it properly interacts with all integrated systems, including, for example, medical information sharing systems. This requires not only expertise in design, but also expertise in testing and quality assurance, regulatory compliance, as well as maintenance and management of the system over its lifecycle. The system must be designed in such a way that it considers the future integration of other systems or the obsolescence of other components with which it is integrated. Finally, and most critically, it must employ safeguards and measures that support the many cases in which a medical professional is required to further assess diagnostic information. While automated systems can be enormously beneficial to the fast and accurate diagnoses of patients, as a result of the numerous risks involved with these systems, a software engineer should be responsible for these projects, to ensure they are being designed in a safe manner.

Conclusion

Medical applications of software engineering are numerous, and center around the secure and accurate measurement, recording, processing, and sharing of patient data. Engineered software in this area has the potential to reduce administrative burdens and increase the speed of diagnoses, but it must be developed according to engineering principles to ensure medical professionals can accurately and reliably conduct their work. As these applications involve ***safeguarding of life, health, and public welfare*** they should be developed and implemented by software engineers.

Appendix D – Transportation: Public transit management systems

Practice Area D: Transportation

Focus: Public transit management system

With public transit being transformed from a system of static, scheduled, fixed routes to dynamic on-demand networks, the demand for real-time measurement and analysis of transit systems has become a necessity. Transit authorities receive real-time information which allows for effective management of fleet resources. These systems can track transit routes and ridership, as well as providing live reporting of maintenance issues, breakdowns, delays, and other incidents. The systems also incorporate safety features such as event-triggered video/audio feeds and push-button calls to emergency services.[6]

Given that disruption of transit services could have widespread impacts of the functioning of a city and its infrastructure, including significant economic impacts and a rise in traffic incidents, systems must be designed in a way that ensures secure, efficient interactions among a wide variety of components (e.g., GPS systems, dispatching services, real-time ridership data, traffic information services, route disruption data sources, safety-event triggers). Software plays a critical role in the analysis and representation of these numerous components, and because of the system's overall complexity, the software's design requires the application of conceptual activities and engineering principles. Software engineers' understanding of all aspects of the software development life cycle, including coding standards, code reviews, source control management, build-processes, testing, and operations should be employed for the safe design of public transit management systems.

Conceptual Activities and Engineering Principles

Whether its in developing algorithms to process real-time information from numerous transit data sources, or developing automated dispatching systems to manage emergencies, software engineers involved with the design and development of PTMSs undertake a wide variety of ***conceptual activities*** and the ***application of engineering principles***, including:

- Designing, prototyping, testing, and implementing complex, highly scalable, reliable systems and web-based applications to ensure that they are fault-tolerant and reliable under a wide variety of use cases;
- Processing, integrating, and representing data from a variety of sources and platforms, including physical systems (e.g. buses, roads), in such a way that takes into consideration and mitigates risks such as faulty instrumentation, tampering, or corrupt data;
- Delivering accurate transit data such as schedules and delays to transit customers, integrating disparate data sources for a variety of use cases while anticipating software-related risks such as Distributed Denial of Service (DDoS) attacks or hacking;
- Acquiring, storing, and managing ridership data (e.g., transit passes, customer accounts) in such a way that it complies with privacy regulation and other laws relating to cybersecurity and data management;
- Modelling complex interactions in a real-time system using efficient data structures that support millions of transactions per minute by users and operators;
- Integrating operating systems, memory management, performance/resource optimizations, database interactions, network programming, concurrency, multithreading, fault tolerance, and the monitoring, security, and operability of the system; and
- Putting in place extensive testing, verification, monitoring, and validation protocols, which requires an advanced understanding of quality assurance principles.

Safeguarding

When designing PTMSs and other transit-related systems, software engineers must often integrate multiple hardware and software systems to ensure their safe and secure interoperation. If these systems are not designed in such a way that accounts for, among other factors, their interaction with other systems, their full life cycle from development to obsolescence, cybersecurity risks, and any potential risks to the public, they can pose substantial danger to the safety of users (including operators) and the functioning of city infrastructure. To ensure their safety, these software systems must be designed in such a way that they utilize the expertise of interdisciplinary teams comprised of planners, transit engineers, operators, and other relevant professionals; a software engineer's knowledge of others' expertise is critical to its safe and proper integration within software systems. For the design of any software components of PTMSs that could involve risks involving ***safeguarding life, health, and public welfare***, such as those outlined here, a software

engineer should be retained.

Example: Public transit management systems

Public transit management systems (PTMSs) are advanced software applications or suites that utilize and integrate multiple applications and data sources to assist in the management of public transit systems. To support the immense task of providing transit information and tracking routes—and in some cases, acting as a transactional platform—PTMSs often integrate multiple management systems such as accounting software, payment processing software, and route scheduling tools. To accomplish these tasks, a PTMS must be designed in such a way that takes into consideration the lifecycle of system components and vulnerabilities to cybersecurity attacks, while remaining reliable and responsive with regard to its intended application. A software engineer can provide a higher standard of care and expectations of a structured and rigorous development process, often including adherence to published development process standards. In short, because of the complexity of a PTMS and the multiple factors involved in its development and management, to be designed safely, it requires ***conceptual activities*** and the ***application of engineering principles.***

In addition to assisting with the management of transit system resources, a PTMS may also integrate features to support incident investigations, such as driver assaults, traffic accidents, or fraudulent activity. These features must be designed in such a way that ensures accurate, reliable, and regulatory-compliant recording and monitoring of user data (e.g. operator behaviour, vehicular performance, registered rider sign-ins). For example, when questions arise regarding a preventable traffic accident, a PTMS will have collected data, such as information regarding the vehicle's geographical location, speed, and driving direction, at regular intervals of time. Because of the risks involved with faulty or improperly managed data, when a PTMS software component incorporates functions involving ***safeguarding of life, health, property, economic interest, public welfare, and the environment***, it should be designed by a software engineer.

Conclusion

Transit systems and city infrastructure are complex entities with an immense number of continuously changing data points. When properly designed, the integrated systems comprising a PTMS can help manage complexity, enhance transportation and safety, and safeguard against the significant problems that can occur in relation to service disruptions. Due to their complexity and due to the risks involved with improperly conceptualized or integrated PTMSs, a software engineer should undertake or oversee their development, and should be involved with any components relating to safeguarding the public.

Appendix E – Letter from Canadian engineering regulators on “Use of ‘Software Engineer’ and Related Titles in Canada”

In July 2022, Engineers Canada and the 12 engineering regulators across Canada co-signed a statement reiterating that the use of titles such as “software engineer”, “computer engineer”, and similar titles that prefix “engineer” within IT-related disciplines and practices are restricted to those who are licensed as an engineer.

Read the statement here.

Bibliography

1. Canadian Engineering Qualifications Board, *Software Engineering Syllabus*, 2004. http://www.engineerscanada.ca/e/pu_syllabus_1.cfm
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. 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
4. Institute of Electrical and Electronic Engineers, *IEEE Standard for Software Safety Plans,* IEEE Standard 1228-1994, New York, NY. http://standards.ieee.org/findstds/standard/1228-1994.html
5. International Atomic Energy Agency, *Verification and validation of software related to nuclear power plant instrumentation and control,* Technical reports series no. 384, Vienna, 1999. http://www.iaea.org/NuclearPower/IandC/index.html
6. 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. http://www.iaea.org/NuclearPower/IandC/index.html
7. Biomedical Engineering Desk Reference, Buddy D. Ratner et al, Academic Press, 2009, ISBN: 9780123746467
8. 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
9. 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
10. 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/).

Footnotes

[1] An exception to this fact is the title “engineer” as it pertains to train operators.

[2] Any member of L’Ordre des ingénieurs du Québec who practices software engineering is nevertheless under the jurisdiction of the regulator.

[3] In Quebec, this only applies when the work has been designed by one holding the engineering title (i.e., ing.).

[4] https://engineerscanada.ca/publications/public-guideline-on-the-code-of-ethics, accessed January 24, 2021.

[5] A well known example of the risks of improper software engineering in medical technology is Therac-25, a computer controlled radiation therapy machine whose flawed software system design periodically led to patients receiving dangerously high, and in some cases lethal, doses of radiation.

https://escholarship.org/uc/item/5dr206s3, accessed June 17, 2022.

[6] For more information on the topic of artificial intelligence engineering technology in autonomous and connected vehicles, please consult Engineers Canada’s national position statement on this topic. A recent example illustrating risks in the area of vehicle automation can be found in case studies of the Boeing 737 Max’s Maneuvering Characteristics Augmentation System (MCAS), whose faulty implementation led to two crashes between 2018 and 2019 and the subsequent grounding of the 737 Max fleet for 20 months.