Open Source Software   
in Government

A Course for Public Sector Executives

**Version 1.0**

Open North

December 1, 2022

**Prepared for:**  
  
The Institute for Citizen-Centred Service

Open Source Software Working Group

# Front Matter

## About This Work

These learning materials were developed by Open North. They were commissioned by the Institute for Citizen-Centred Service on behalf of the Public Sector Service Delivery Council and the Public Sector Chief Information Officer Council (Joint Councils).

**Primary authors:** Matthew Claudel (Field States), Steven Coutts (Open North)

**Contributors:** Ayesha Zamudio-Vazquez (Open North), Merlin Chatwin (Open North), John Griffin (Open North), Elena Findley-de Regt (Foundation for Public Code), Ben Cerveny (Foundation for Public Code)

**We would also like to acknowledge the contributions of the Open Source Software Community of Practice:** Guillaume Charest (Canadian Digital Service), Alexandre Cloutier (Canada Revenue Agency), Olivia Courney (Ontario Digital Service), Kevan Hannah (Ontario Digital Service), Alan Harnum (Ontario Digital Service), Katy Lalonde (Ontario Digital Service), Patrick McCabe (Nova Scotia Digital Service), and Bianca Tomazeli (Ville de Montréal)

**Citation:** Open North. “Open Source Software in Government: A Course for Public Sector Executives – v1.0.” Institute for Citizen-Centred Service, 2022.

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) License](https://creativecommons.org/licenses/by-sa/4.0/legalcode).

If you create an adaptation of this work, please add the following disclaimer along with the attribution:

*This is an adaptation of an original work by the Institute for Citizen-Centred Service and Open North. Views and opinions expressed in the adaptation are the sole responsibility of the author or authors of the adaptation and are not endorsed by the Institute for Citizen-Centred Service or Open North.*



# Table of Contents

[Front Matter ii](#_Toc120030484)

[About This Work ii](#_Toc120030485)

[Table of Contents iii](#_Toc120030486)

[Course Overview 1](#_Toc120030487)

[Course Learning Objectives 1](#_Toc120030488)

[Course Module Overview and Structure 1](#_Toc120030489)

[Module 1: Introduction to Open Source Software 3](#_Toc120030490)

[Learning Objectives 3](#_Toc120030491)

[Key Points 3](#_Toc120030492)

[Section 1: How Do Governments Use Software? 5](#_Toc120030493)

[Section 2: Software Basics 6](#_Toc120030494)

[Section 3: Benefits and Challenges of Open Source Software in Government 11](#_Toc120030495)

[Additional Resources 14](#_Toc120030496)

[Module 2: Building an Open Source Enabling Environment 15](#_Toc120030497)

[Learning Objectives 15](#_Toc120030498)

[Key Points 15](#_Toc120030499)

[Section 1: Policy and Legislative Landscape 17](#_Toc120030500)

[Section 2: Key Issues and Concepts in Building an Open Source Ecosystem 20](#_Toc120030501)

[Section 3: Strategies for Building an Open Source Ecosystem 23](#_Toc120030502)

[Additional Resources 27](#_Toc120030503)

[Module 3: Procurement, Contracting and Budgeting for Open Source Software 28](#_Toc120030504)

[Learning Objectives 28](#_Toc120030505)

[Key Points 28](#_Toc120030506)

[Section 1: Software Licensing, Revenue and Development Models 29](#_Toc120030507)

[Section 2: Key Issues and Concepts in Digitally-Native Procurement 33](#_Toc120030508)

[Section 3: Contracting and Budgeting for Open Source Software 39](#_Toc120030509)

[Additional Resources 43](#_Toc120030510)

[Module 4: Open Source Software Security and Maintenance 45](#_Toc120030511)

[Learning Objectives 45](#_Toc120030512)

[Key Points 45](#_Toc120030513)

[Section 1: Introduction to Open Source Software Security and Maintenance 46](#_Toc120030514)

[Section 2: Key Issues and Concepts in Maintaining and Securing Open Source Software 48](#_Toc120030515)

[Section 3: Strategies for Maintaining and Securing Open Source Software 52](#_Toc120030516)

[Additional Resources 54](#_Toc120030517)

[Module 5: Collaborating on Open Source Software 55](#_Toc120030518)

[Learning Objectives 55](#_Toc120030519)

[Key Points 55](#_Toc120030520)

[Section 1: What Does It Mean to Work with Open Source Software across Jurisdictions? 56](#_Toc120030521)

[Section 2: Strategies for Collaborating Across Jurisdictions on Open Source Software 60](#_Toc120030522)

[Additional Resources 63](#_Toc120030523)

[Glossary 64](#_Toc120030524)

[Appendix A: Executive One-Pagers 69](#_Toc120030525)

[Executive One-Pager #1: Why Open Source Software? 70](#_Toc120030526)

[Executive One-Pager #2: Building an Open Source Enabling Environment 72](#_Toc120030529)

[Executive One-Pager #3: Procurement, Contracting and Budgeting for Open Source Software 74](#_Toc120030532)

[Executive One-Pager #4: Open Source Software Security and Maintenance 75](#_Toc120030535)

[Executive One-Pager #5: Collaborating on Open Source Software 76](#_Toc120030538)

[Appendix B: Implementing the Course 77](#_Toc120030541)

# 

# Course Overview

Public sector executives and senior managers have an opportunity to lead digital transformation of mission-critical service delivery and operations within their organizations by embracing open source software solutions. To drive this transformation, they need to:

* Re-orient organizational culture to become open source software friendly
* Invest in staff capacity building
* Champion digitally-native approaches to procurement, budgeting and contracting that support open source software adoption
* Understand the fundamental differences between conventional proprietary software and open source software
* Leverage the potential of cross-jurisdictional collaboration to build and support open source software applications

The purpose of this course is to equip public sector executives and their colleagues with the ideas, tools and resources they need to overcome the challenges that are often associated with using open source software – and to take full advantage of its benefits. This course has been designed primarily from the perspective of non-technical executives, directors and departmental leaders, but will provide important insights for technical staff as well.

## Course Learning Objectives

By taking this course, learners will:

1. Develop basic knowledge of open source software fundamentals, how it differs from conventional proprietary software, and what value open source software brings in a public sector context.
2. Learn to recognize the common barriers to using open source software in government, and identify strategies for overcoming them.
3. Understand how an applied, operational framework for policy and management decisions can enable a well-balanced approach to software – whether open source or conventional proprietary.

## Course Module Overview and Structure

The content has been divided into five modules that each address specific aspects of enabling, working with, and benefitting from open source software in government:

* Module 1, Module 2, and Module 5 provide a high-level overview of open source software and how it is different from proprietary software, organizational factors that can enable or hinder its adoption and use in government, and what it means to participate in the broader open source software ecosystem.
* Module 3 provides detailed information on procurement, contracting and budgeting considerations related to open source software. The contents of this module will be of interest to government staff working in **procurement, finance** or **legal** capacities.
* Module 4 provides detailed information on various security and maintenance aspects associated with open source software.

Each module contains Key Points, sub-topics and their associated challenges, strategies, and further resources for those interested in learning more.

Also accompanying each module is an **Executive One-Pager** summarizing the top-level message and key points from each module. Staff can use these documents to communicate key messages to their senior or executive-level management and help to make a case for – and implement – open source software. The courseware will inform decisions about integrating and aligning the use of open source software with organization-wide priorities and commitments.

# Module 1: Introduction to Open Source Software

## Learning Objectives

In this module we will:

* Review the general use cases for software in government and learn how government software is unique.
* Identify the main differences between open source software and proprietary software by exploring the origins and motivations of the open source software movement, as well as the fundamentals of how open source software works today.
* Introduce the fundamentals of obtaining and maintaining open source software, highlight the benefits of using open source software in government, identify the unique challenges associated with it, and introduce strategies for overcoming those challenges – strategies that will be elaborated in greater detail throughout the course.

## Key Points

* **Governments and public sector agencies across Canada and worldwide are already using open source software in significant ways.** As governments at all levels incorporate software into a wide variety of processes and services to meet their objectives – from efficient public service delivery to social and environmental goals – open source software presents an excellent option for ensuring quality, ongoing control, and resource efficiency.
* **Open source software is fundamentally different from conventional proprietary software** in several ways: how it is licensed, owned and managed; how it is developed and maintained; and how it is financed and commercialized. Open source software is freely distributed under an open-source license. Many open source software projects are maintained by a global community of users and developers, while others have a smaller contributor base. While the software itself is free, your organization may invest in in-house staff capacity to integrate and maintain software, or contract with a vendor to add specific features or do a custom integration with your existing digital environment.
* **Open source software does not have to be built from scratch.** Hundreds of full open source software projects and even more small component open source modules offer as many options and functionalities as conventional proprietary software. These open source projects and modules serve as a foundation upon which new open source software can be built.
* **Open source software has the potential to bring a variety of unique benefits to government** – including customizability, interoperability, and transparency. While there are barriers to adopting open source software in government – including a lack of in-house technical capacity and inflexible procurement practices – fortunately, there are a variety of strategies and best practices that can help you to overcome these barriers.

## Section 1: How Do Governments Use Software?

### Software in Government: Use Cases

Governments use software for a wide range of administrative, financial, and operational purposes, from public-facing services to procurement management to secure email. These use cases include (but are not limited to):

* Direct service provision to citizens (issuing permits and licenses, civic issue tracking)
* Fee and payment processing (property taxes, fines, etc.)
* Business intelligence (tracking performance of business units, programs, and services)
* Data management (open data portal)
* Human resources management (payroll and benefits administration, talent acquisition)
* Asset management (monitoring equipment and infrastructure throughout its life cycle, modeling risk and cost scenarios)
* Procurement management (issuing tenders, managing vendor relations)
* Communications (content management, agenda and meeting management, emergency alert services)

### How Government Software is Unique

The government context presents unique requirements for software. The process of obtaining and maintaining software – and the design of the software itself – has distinctive features in public sector environments. Government requires software that:

* Is highly reliable
* Is secure enough to store sensitive or personally-identifiable information
* Can be sourced according to government procurement laws
* Can be budgeted for up front, typically years in advance (as with government purchases in general)
* Can be integrated into legacy IT systems and digital environments
* Will comply with regulations and standards specific to a jurisdiction, such as data residency requirements (i.e., hosting data on local servers as opposed to cloud-based storage outside the jurisdiction) or complying with language and accessibility requirements

## Section 2: Software Basics

### Software Basics: Key Concepts and Terminology

Throughout this course, we will discuss various aspects of buying, using and maintaining software, with particular attention to the public sector context. A handful of fundamental concepts underpin this discussion:

* **Development** refers to *how software is built.* Software consists of lines of code. Software can be built from scratch, but more often, software is composed of pre-existing modules (which perform specific functions, such as sign-in authentication) and those modules may, themselves, be open source. For the purposes of this course, development also encompasses the ongoing updating and maintenance of software.
* **Integration** refers to implementation in a particular context. Software rarely works right “out of the box.” Depending on your organization’s needs, it will require custom features and will always need to be integrated with your existing digital environment.
* **Distribution** refers to *how software gets to the end user*. This includes the legal software use license (proprietary or open source) as well as the method of distribution (cloud, hybrid, or on-premises).
* **Revenue** refers to *who pays who for what*. This might be Software-as-a-Service (SaaS), an on-premises deployment of proprietary software, or fee-for-service to create a custom implementation of open source software.
* **Clients, Vendors, Contractors and Users** comprise *the major stakeholders in a software ecosystem*. The relationships between these stakeholders emerge from Development, Distribution, and Revenue models.

### Software Licensing Models

Anyone can develop software code and apply whatever license they choose. A license determines who can use the software, how they can use it, if they can alter it, and so on. There are two primary software licensing models:

* **Proprietary** licensing refers to software for which the source code is legally owned by the individual or company that created it. That entity has the sole and exclusive right to edit and commercialize the code as they see fit.
  + Examples: Microsoft Windows and Apple OS X (operating systems), Microsoft Office (productivity suite), Adobe Creative Suite (design software), SAP ERP (enterprise resource planning software), ArcGIS (geographic information system).
* **Open source** licensing refers to software that is available for anyone to access, use or change the source code. The source code is made available online and contributions are managed using a platform like GitHub.
  + Examples: Linux (operating system), Firefox (web browser), Python (programming language), Open Office (productivity suite), Thunderbird (mail client), Apache (web server), QGIS (geographic information system).

### How Proprietary Software Works

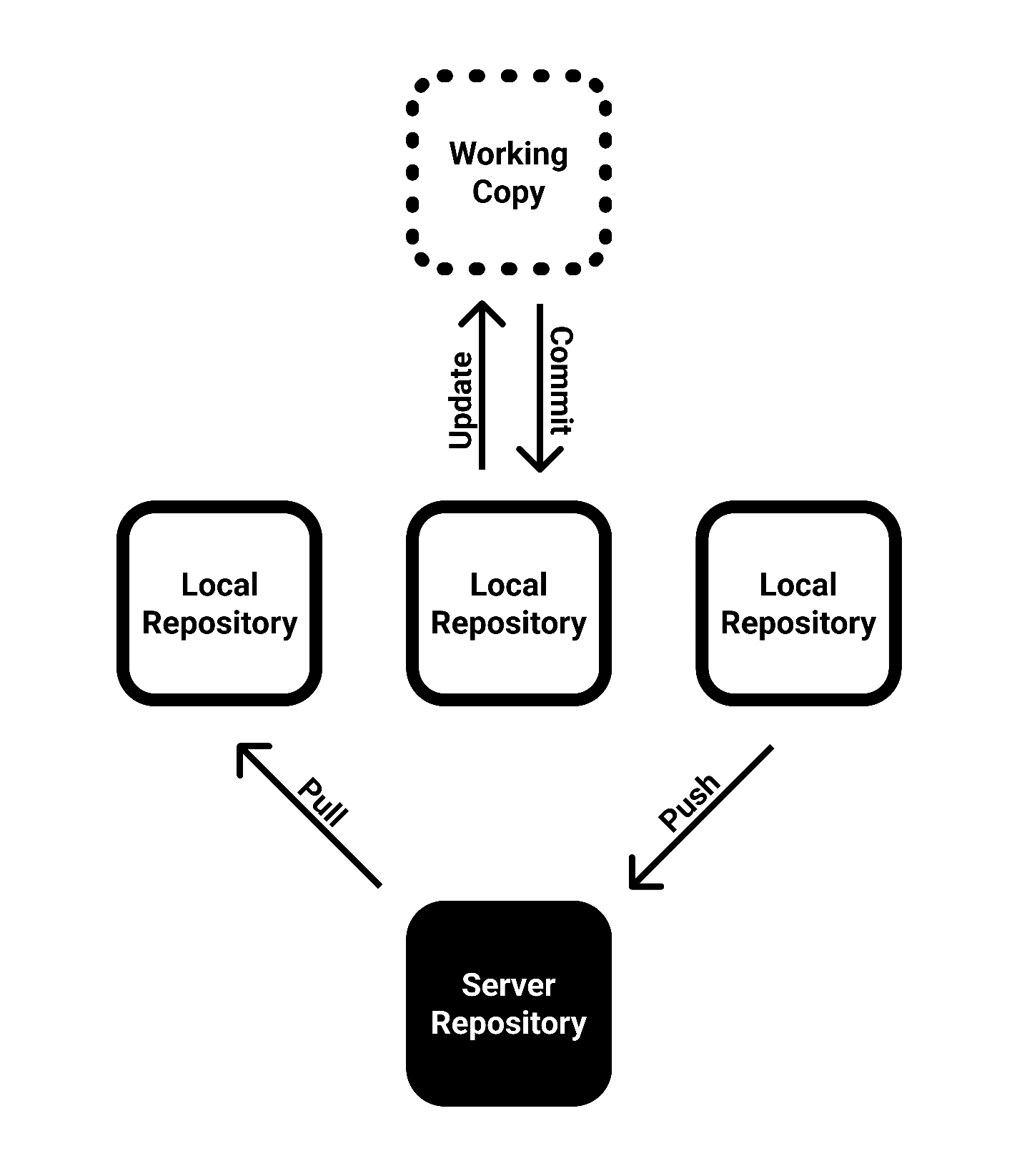
More than likely you already use proprietary software on a daily basis and have a basic understanding of how it works. The following is a summary of key stages along the proprietary software value chain.

* **Development**: Software vendors build and commercialize software. Vendors typically provide core updates (or “patches”) to software so that it remains bug-free and compatible with changing digital environments (updates may be free or paid).
* **Distribution**: The organization that built the software maintains ownership of the rights to access, use, and commercialize the source code through a legally enforceable license. Software may be hosted on the cloud (online), with users paying to access it, or hosted on-premises (on local servers).
* **Integration:** Software companies typically offer support but may charge fees. Some software vendors will provide custom integrations with your organization’s existing digital environment; if not, you will need to contract another service provider.
* **Revenue**: End users pay to use the software, according to a revenue model (flat rate, usage-based, tiered, freemium, etc.). Software companies may issue patches, new features, upgrades and integrations that are included in the fee structure, or for which end users pay an additional fee.

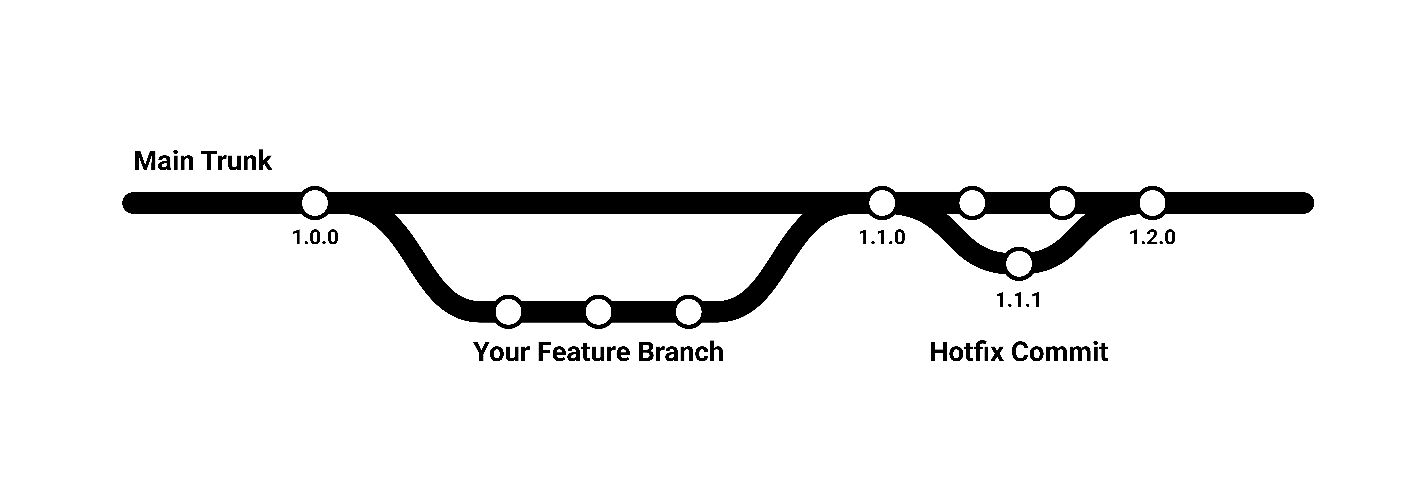
### How Open Source Software Works

Open source software works in a fundamentally different way than proprietary software.

* **Development:** A developer writes the first version of a piece of open source software and uploads it to a public online repository under an open source license. The code is then maintained using a **distributed version control system** (Git being the most popular) in which each user has a working copy and full change history of a given codebase (see Figure 1.1). GitHub, the largest hosting service for these repositories, provides a reliable environment in which a global community can manage, harmonize and reconcile simultaneous (possibly conflicting) versions of the software, identify and fix bugs, and streamline the process of editing.
  + There are several models of open source software development, each of which has slightly different approaches to how merges with the main codebase are managed and how frequently they happen. For the most part, these models follow a branching structure, in which the main codebase functions as a ‘trunk’ with ‘branches’ created as individuals or organizations create copies – or ‘forks’ – and add their own features (see Figure 1.2). In this way, open source software is continually improving through contributions from an open community.
  + Some open source software has a governance structure in place to review and validate any changes to the source code. This ensures the utility and integrity of contributions. Validated changes are integrated back into the main trunk of the tree.



**Figure 1.1:** This diagram shows in a simplified way how a distributed version control system works. A main server repository stores all of the files associated with an open source project. Each organization collaborating on the project maintains a local repository which is a copy of the main repository. Every programmer maintains a working copy of their respective local repository (which in turn is a copy of the main repository) on their hard drive. Programmers can commit changes to and update their local repository without any interference. They can update their local repositories with new data from the main server repository by an operation called “pull” and make changes to the main repository by an operation called “push” from their local repository.



**Figure 1.2**: A simple branching development model for open source software. The main trunk represents the ‘original’ source code. Any individual or organization can create a copy – or ‘fork’ – and add their own features. “Hotfix” branches are used to quickly patch production releases.

* **Distribution:** An open source license grants anyone the rights to use, study, change, and distribute the software and its source code to anyone and for any purpose (often with attribution).
  + Regardless of how software is developed, it can be released under an open source license. The challenges and opportunities of open source software are more specific and unique once it has been released under an open source license, entering the maintenance and adaptation phase.
* **Integration:** At any point, someone may decide that their needs are unique enough to merit a fully customized version of the software. They can create a copy of the source code and alter it as they see fit, without re-integrating changes into the main source code. This is known as a “hard fork” because it starts a new path, disconnected from the original source code.
* **Revenue:** End users do not pay for access to the source code. However, that doesn’t mean open source software is free – there are costs associated with adapting, hosting, and maintaining the software. These costs might go toward in-house personnel or contracting an external technical service provider.

#### Box 1.1: Origins of the Open Source Software Movement

The open source software movement was inspired by the opportunity to bring together a global community to create, grow and adapt software freely and collaboratively.

* “Open source” software is not the same as “free” software. In English, “free” can mean “at no monetary cost” (gratis) and/or “with little or no restriction” (libre).
* In the 1990s, software developers started making their software free (gratis) because the marginal cost of each unit was zero. The software was given away at no cost in this paradigm, but the rights stayed with the original developer.
* The open-source software movement emerged soon after. It was important because it focused on opening up the rights to access and edit code, not just making software free to use. Software would be free (libre) – so that other developers could not only use it but also change and improve it.
* In an iconic essay, one of the fathers of the Open Source movement, Lawrence Lessig, coined the phrase, “think of free speech versus free beer”2: free software is like free beer (no cost), while open source software is like free speech (unrestricted access and reproduction).

1 Eric S. Raymond, “The Cathedral and the Bazaar,” *Knowledge, Technology & Policy* 12, no. 3 (September 1999): 23–49, <https://doi.org/10.1007/s12130-999-1026-0>.

2 Lawrence Lessig, “Free, as in Beer,” *WIRED*, September 1, 2006, <https://www.wired.com/2006/09/free-as-in-beer/>.

## Section 3: Benefits and Challenges of Open Source Software in Government

### The Unique Benefits of Open Source Software in Government

There are nine factors that make open source software ideal for use in a government context:

1. **Openness and transparency**. Open source software is just that: open. It isn’t a black box. Global and local communities are able to audit the code, evaluate the implications of its design, and hold each other accountable.
2. **Increased interoperability**. Open source software tends to be built from open source modules. That means any given piece of open source software is likely to use the same sub-modules (or compatible sub-modules) as another. With proprietary software, there is a financial incentive for vendors to create dependencies with their own or their partners’ proprietary software. With open source software, there is an incentive for interoperability because it increases efficiency and replication.
3. **Decentralized, community-led product support**. Open source software relies on different mechanisms for updating and maintaining software, finding bugs and fixing them, but is ultimately just as secure as proprietary software. Because the code is open, security auditors and contributors can inspect the code for security risks or flaws. Open source software projects supported by active communities receive constant updates, which ensures performance and cybersecurity in changing conditions.
4. **Controlled costs**. Open source software itself is free, and you have direct control over all associated costs – whether you hire in-house staff or contract with external service providers. There are no hidden fees, expensive maintenance, paid features, or contract renegotiations.
5. **Strengthened local tech economy**. Local software contractors are well suited to build custom features or integrations for government organizations, because they develop long-term relationships and understand local priorities. In contrast, proprietary software is generally created and commercialized by monolithic software vendors that do not have any connection to local context. By hiring local firms, governments can strengthen the local economy, particularly its tech sector.
6. **Freedom from vendor lock-in**. Proprietary software can be operationally or contractually entrenched in an organization, putting the government at a disadvantage when making decisions in the future. Proprietary software vendors have an incentive to create dependencies, which makes it challenging to freely and fairly choose the best solution. Open source software creates open markets for providers of all kinds of support.[[1]](#footnote-1)
7. **Adaptable to changing requirements**. Policy and regulation changes constantly (for example, accessibility standards, or protocols for personally identifiable information). Proprietary software vendors may or may not release updates to meet changing requirements. Updates are costly and may require renegotiation of contracts. Open source software, on the other hand, can be quickly adapted and shared across jurisdictions.
8. **Customizable to local context and use cases**. While many government organizations perform similar functions, the way in which those functions are carried out may vary from place to place. Governments can address very niche local use cases by developing their own open source software or by banding together with other governments to share a core set of open source software modules and each pay a small fee for a local contractor to create custom features for their unique environment.
9. **Reduced liability**. Government organizations can use open source software however they want (subject to applicable laws and policies), with no additional obligations or restrictions imposed by a vendor. Furthermore, open source software is not subject to the legal risks associated with proprietary solutions, especially those that arise from contracts.

### The Challenges of Using Open Source Software in Government

There are five primary challenges that make it difficult to use open source software in a public sector context. Once they’re named, these can be overcome – and this course will empower you with strategies and tools.

1. **Procurement and budgeting**. Open source software has a very different cost structure. The software itself is free, but there are costs associated with customization and integration. This means a shift from buying a product to building a solution. There is often a lack of funding for developing new software, and resistance to subsequently offering it for free to others. Procurement norms, contract templates and approved vendor lists create barriers to obtaining and using open source software or contracting open source software service providers.
2. **In-house expertise**. The education and training most public sector professionals receive still prioritizes proprietary software, and computer systems often come pre-loaded with proprietary software. For these, and many other reasons, public sector agencies conventionally rely on off-the-shelf software from major vendors, as opposed to hiring and retaining in-house expertise. Staff are often traditionally trained in using proprietary software so the introduction of new open source solutions may require retraining.
3. **Supplier availability**. Smaller municipalities and rural areas may not have a mature local ecosystem of technology companies that are capable of providing services. And while there are several agencies that work across Canada, serving a broad swath of customers, identifying one that can provide services that meet your organization’s needs can be a challenge.
4. **Rate of change**. Because open source software is often developed and maintained by a global community, it may have faster or slower rates of updating and change. If updates you need are not emerging from the community, your organization may need to proactively make those changes, or hire a firm that can. When considering open source software, it is important to evaluate the level of activity within the support community and develop a contingency plan in case the open source software is no longer adequately supported and you need to find an alternative.
5. **Governance**. Open source software requires active participation involving contributors and users from different organizations, which can create challenges in managing the responsibility for maintenance.

### Summary

This first module has introduced you to the basics of open source software. You should have a better understanding of the functionality and history of open source software, as well as the specific benefits and challenges of using open source software in a government context.

The following modules will give you a closer look at several different aspects of obtaining, using, and maintaining open source software in government. They will equip you with tools and strategies for doing that work well.

[Module 2: Building an Open Source Ecosystem](#_Module_2:_Building)

[Module 3: Procurement, Contracting & Budgeting](#_Module_3:_Procurement,)

[Module 4: Open Source Software Security & Maintenance](#_Module_4:_Open)

[Module 5: Collaborating on Open Source Software](#_Module_5:_Collaborating)

## Additional Resources

[**Who’s using GitHub?**](https://government.github.com/community/)**:** A list of hundreds of government agencies at the national, subnational, and municipal levels who are using GitHub to share and collaborate on open source projects.

[**Linux Professional Institute (LPI)**](https://www.lpi.org/) is a nonprofit organization that provides training and certification, and supports open source software communities.

[**The Government of Canada Digital Standards: Playbook**](https://www.canada.ca/en/government/system/digital-government/government-canada-digital-standards.html) “form[s] the foundation of the government’s shift to becoming more agile, open, and user-focused. [It] will guide teams in designing digital services in a way that best serves Canadians.” This is a great place to get started on your journey with open source software.

[**The Open Resource Exchange**](https://code.open.canada.ca/en/index.html) features a searchable catalog of open source software, as well as open source code projects, open standards, partnerships and open designs in use by different levels of public administrations in Canada.

[**Open Source Basics (Intel Software)**](https://www.youtube.com/watch?v=Tyd0FO0tko8): An introductory video explaining the basics of open source software.

[**What is Open Source explained in LEGO (Socialsquare)**](https://www.youtube.com/watch?v=a8fHgx9mE5U): Another introductory video explaining the basics of open source software (this time using LEGO).

[**The mind behind Linux: Linus Torvalds (TED)**](https://www.youtube.com/watch?v=o8NPllzkFhE): An interview with Linus Torvalds, who transformed technology twice – first with the Linux kernel, which helps power the Internet, and again with Git, the source code management system used by developers worldwide.

# Module 2: Building an Open Source Enabling Environment

## Learning Objectives

In this module we will:

* Identify the key enabling factors and barriers to the adoption of open source software, related to staffing and organizational structure.
* Discuss the wide variety of job functions, professional profiles and skill sets involved in developing, implementing and maintaining open source software projects in a government organization.
* Explore best practices for building and leading a right-sized technical team.

## Key Points

* **A growing policy and legislative landscape supports the adoption of open source software.** However, current structures, practices, and mentalities within many government and public sector agencies can create barriers when it comes to open source software adoption.
* **Technical staffing is one of the greatest perceived barriers to open source software adoption.** A lack of in-house capacity and collaborative culture can lead to over-reliance on vendors: Few government organizations have full-time staff (at leadership and implementation levels) with necessary skills and knowledge to procure, build, and/or maintain open source software. As a result, governments typically rely on vendors.[[2]](#footnote-2) This can cause vendor lock-in, ballooning costs and hidden fees, and loss of control (over functionalities, updates and compliance).[[3]](#footnote-3)
* **Executives in organizations of all sizes can overcome these challenges by making strategic investments, reducing real and perceived risk, and encouraging a cultural shift in the organization**. This includes:
  + Learning how open source software procurement, development, and maintenance works and supporting internal working groups and champions.
  + Driving an organizational culture that encourages working in the open and continuous user engagement.
  + Engaging with a variety of partners, peer organizations and service providers to complement the internal team.

## Section 1: Policy and Legislative Landscape

There is a growing policy and legislative landscape addressing the use of software at the federal, provincial/territorial, and municipal levels. Some of these policies and laws directly specify the use of open source software, while others indirectly affect the use of open source software.

### Open Source at the Federal Level

There are numerous policies at the federal level that set out the Government of Canada’s approach to open source software.

* The Government of Canada is “mandated to release its own source codes under Open Source Licenses as long as it is compatible with core administrative law principles such as transparency, accountability, legality and procedural fairness.”[[4]](#footnote-4)
* In 2018, the Government of Canada adopted the [Open Data Charter](https://opendatacharter.net/principles/) as a demonstration of its commitment to open data. While it is not a legally binding instrument, the Charter and its “open by default” principle lay the groundwork for the use and development of open source software.
* Additional guidance for government staff at the federal level includes the [Directive on Service and Digital](https://www.tbs-sct.canada.ca/pol/doc-eng.aspx?id=32601) and the [Government of Canada Digital Standards: Playbook](https://www.canada.ca/en/government/system/digital-government/government-canada-digital-standards.html).

### Open Source at the Provincial Level

Provinces have a significant degree of control over the specific software and services they use, as well as the interpretation and implementation of federal policies, in the form of provincial data policy.

* The [Ontario Digital Service Standard](https://www.ontario.ca/page/digital-service-standard) directs provincial ministries and agencies to "use open standards, open source software and common government platforms where available” and to "favour open tools that are accessible and have strong developer community support.”[[5]](#footnote-5)
* The Province of British Columbia has published [Guidelines on the Use of Open Source Software](https://www2.gov.bc.ca/assets/gov/government/services-for-government-and-broader-public-sector/information-technology-services/standards-files/1_02-v1_0-guidelines_on_the_use_of_open_source_software.pdf) as well as a [Policy Framework](https://github.com/bcgov/BC-Policy-Framework-For-GitHub) guiding employees’ use of GitHub.
* The Province of Quebec “advocates the use of free software when it is the best choice and will ensure that leaders have the necessary support for this purpose.” Its Open Source Software information resource includes a Reference Guide, Maturity Analysis Guide and Total Cost of Ownership Analysis Guide.[[6]](#footnote-6) Quebec has also developed the [Québec Free License](https://forge.gouv.qc.ca/licence/en/) – an Open Source Software license validated by the Open Source Initiative and specifically designed for use in the public sector, with three degrees of restriction on use, application and modification.[[7]](#footnote-7)

Even where there is no explicit province-wide policy promoting the use of open source software, several provinces have developed open source projects to meet specific needs. For example, the Government of Alberta has released an [open source toolkit](https://github.com/abgov/ab-compensation-transparency-toolkit) to support disclosures under Alberta's Public Sector Compensation Transparency Act. During the height of the COVID-19 pandemic, the Government of Nova Scotia released an [open source tool to help citizens self-assess when to call the 8-1-1 non-urgent health services hotline](https://github.com/Nova-Scotia-Digital-Service/when-to-call-811).

### Open Source at the Municipal Level

Municipalities also have a significant degree of control over the specific software and services they use, and the primary lever is procurement. Staff should be familiar with the range of open source software in existence – and a good place to start is knowing how other jurisdictions have addressed similar needs.

* [Several Canadian municipalities](https://cityssm.github.io/municipal-github-rankings/) are active in developing and sharing open source projects on GitHub.
* Some municipalities have adopted policies that explicitly promote the use of open source software (e.g., [Montreal](https://github.com/VilledeMontreal/politique-libre/blob/master/Politique/PolitiqueDuLibre.md#english-version), [Vancouver](https://council.vancouver.ca/20090521/documents/csb5.pdf), [Toronto](https://www.toronto.ca/wp-content/uploads/2022/03/9728-DISFAcc2.pdf)).
* The City of Montreal also maintains an [inventory of open source software it uses](https://donnees.montreal.ca/ville-de-montreal/solutions-en-logiciels-libres).

As at the provincial level, the absence of open source software policies has not stopped several municipalities from developing open source projects to serve local use cases. The City of Sault Ste. Marie, which has been sharing its code via GitHub since 2017,[[8]](#footnote-8) has developed [several open source tools](https://cityssm.github.io/) to support day-to-day municipal operations across departments including a Contract Expiration Tracker, General License Manager, and Parking Ticket System.

### Privacy Legislation

In Canada, privacy legislation at the federal and provincial levels may impact how governments adopt open source software if there is any chance it will handle any personally-identifiable information about citizens. The legislation that applies to a given piece of open source software will depend on who is using the software and what they are using it for.[[9]](#footnote-9)

Legislation enacted outside Canada may only have limited direct impact on the government open source activities in Canada, but it is nonetheless very important to consider in terms of understanding norms around how software is developed for global markets and how personal information is protected within software. For example, the European Union’s (EU) [*General Data Protection Regulation*](https://gdpr-info.eu/) (GDPR) imposes additional responsibilities on data controllers and data processors – these responsibilities may apply outside the EU in cases where data belonging to EU citizens is involved.

## Section 2: Key Issues and Concepts in Building an Open Source Ecosystem

### An Organization that Enables Open Source

Open source software requires that individuals across different governmental business units, departments and agencies build new skills and adapt their day to day role in order to effectively support open source software. This includes individuals working in digital services, IT, any number of direct service delivery departments (e.g. traffic and parking), procurement, budgeting or accounting, legal, and other functions across and within organizations.

Everyone involved needs to be able to: ask the right questions, identify the right outcomes, and have a basic knowledge of the fundamental principles of modern software design.[[10]](#footnote-10)

### In-House Skills and Expertise

Technical staffing is one of the greatest perceived barriers to open source software adoption. Because open source software requires active involvement from the user (as opposed to proprietary software, which is serviced and maintained by the vendor), many assume that it is inaccessible without software development capabilities.

Conversely, many assume that obtaining proprietary software from a vendor means they don’t need technical experts on staff. Without in-house expertise, however, the processes of contracting, procuring and implementing proprietary software can have harmful outcomes. Without adequate knowledge of data ownership and monetization, a proprietary software contract can end up disempowering the public sector in the long term (see [Module 3](#_heading=h.o24g9bwi3uvb)).

Clients – government in particular – rely on software vendors for ongoing support. In the case of proprietary software, there is no choice. The source code is hidden and proprietary, so the vendor is the only party with access. In the case of open source software, anyone can contribute, but there is no single vendor. Governments need to have capacity in-house, collaborate with peers, or hire a provider that can offer support.

### Staff Roles to Support Open Source Software

It is possible to undertake open source software projects without formalized roles. However, this approach may not be sustainable in the long run if your organization’s open source ‘champions’ become unable to drive open source software efforts off the side of their desks in addition to their formal job requirements.

Ideally, you will be able to draw on the skills and expertise of several different roles when embarking on an open source software initiative (see Table 2.1). These roles step into the foreground or background, depending on the specific phase – and once again, not all of these roles need to be in-house or full-time.

Table 2.1: Roles involved in each phase of an open source software initiative

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Involvement by Phase** | | | | |
| **Role** | **Capacity-Building** | **Discovery** | **Procurement** | **Integration** | **Maintenance** |
| **Policy Lead / Senior Executive** | Yes | Yes |  |  |  |
| **Technical Lead\*** | Yes |  | Yes |  | Yes |
| **Product Owner** | Yes | Yes | Yes | Yes |  |
| **Software Developer\*** |  |  | Yes |  | Yes |
| **Procurement Specialist** |  |  | Yes |  |  |
| **Finance Specialist** |  |  | Yes |  |  |
| **Contracting / Legal Specialist** |  |  | Yes |  |  |
| **IT Specialist** |  | Yes | Yes | Yes | Yes |
| **End User Department Head** |  |  | Yes | Yes |  |
| **End User / Departmental Specialist** |  | Yes |  | Yes | Yes |

\* Indicates a role that can exist outside of your organization (e.g. within a peer jurisdiction or contracted through a vendor)

### Breaking Down Departmental Silos

A systemic lack of collaboration between departments and business units can impact open source software adoption. This can happen for a couple of reasons:

* Departments may have divergent or conflicting agendas, which can cause misalignment in priorities and friction before and during the software procurement and development process.
* Software often falls between existing business unit siloes (for example legal, IT, service delivery, budgeting and procurement), which can make it difficult to determine who actually *owns* a process. This is especially true with new or unconventional processes.

Transitioning to cross-departmental protocols and standards requires process change management that is independent of the implicated departments.

Many large private sector firms have established Open Source Program Offices (OSPOs) as a clearinghouse for all things open source within the organization. This can include: “developing collaborations with foundations/organisations and OSS communities, ensuring legal compliance, developing and implementing OSS strategies, launching new projects, providing training, providing guidance to employees on how they could engage in OSS activities and others.”[[11]](#footnote-11)

Some governments and public sector institutions are now adopting the same approach, including the European Commission which created its Open Source Program Office in October 2020. Ultimately, the value of an organizational structure like an OSPO is to move governments beyond simple enactment of high-level policies and towards ensuring “their enforcement and maintenance in a more horizontal, strategic manner coupled with ensuring [their] implementation in a daily practice.” [[12]](#footnote-12)

## Section 3: Strategies for Building an Open Source Ecosystem

### Demonstrate Leadership on Open Source

Strong leadership is essential in any organization – especially government – in setting policies and building up capacity that will ensure the success of open source adoption. If you are taking this course, you are in a great position to demonstrate such leadership.

Capacity building to support the adoption of open source software can start long before a procurement or development process begins. Consider hosting regular, organization-wide workshops on open source software, IT strategy, or procurement innovation to help demystify these topics for everyone. Emphasize opportunities for acquiring better products and services, simplicity, efficiency, and creativity. You will be surprised at the number of people who get excited and ask to be involved.

Reduce barriers to open source adoption within your organization by “giv[ing] teams that want to use open source the permissions they need to do so, and simplify the processes to let them focus on finding and matching software to the needs of their users.”[[13]](#footnote-13)

### Build a Right-Sized Team

Governments – especially small and medium-sized jurisdictions – do not necessarily need to have software developers on staff. However, to effectively obtain and manage digital assets, they should:

* Ensure that a wide variety of staff members have a basic understanding of how software procurement, development and maintenance works.
* Support internal working groups and champions.
* Join networks with peer jurisdictions and open source software communities to share challenges, opportunities, capacities, resources, and processes.

In smaller governments or agencies, the person accountable for the organization’s technology strategy may be best positioned to oversee open source software adoption.[[14]](#footnote-14) It is also important to note that one staff member can perform multiple roles throughout the process. For example, an IT specialist might also be the product owner and the end user.

Beware, however, of relying on passionate “champions” to take on large amounts of open source software work off the side of their desk. Open source software requires active engagement – it can flatline if a key champion burns out, encounters too many barriers, or leaves the organization. If you’re still taking this course, that champion might be *you.* Advocate for long-term, systematic support for open source. The return on investment is significant, especially as more software products are adopted and as procurement, implementation and maintenance processes are solidified.

In larger governments or agencies with substantial internal capacity, setting up a centralized office or team is a good step toward institutionalizing open source.[[15]](#footnote-15)

### Attract Talent by Building an Open Source-Friendly Organization

There are several reasons why hiring and retaining talented IT professionals is easier in an organization that is open source first.[[16]](#footnote-16)

* Open source software is widely used by companies, from development to production. That means talent focus on developing capacity to use it, and seek opportunities to enhance their working knowledge for career advancement;
* By contributing to open source software projects, individuals become part of a public “network of trust” and add to their resumes;
* Some report that working with open source software improves their job satisfaction because they are contributing to a massive community project (source code fixes, bug reports, documentation updates, etc.);
* Open source software provides learning opportunities, because contributors have access to “everything running under the hood” and can learn from more experienced developers).

### Create a Product Owner Role

The product owner is the central figure in an open source software project – whether identifying the need for software, running a procurement or developing in-house. This person has analytical, user research and design thinking skills, and guides the software process, providing a single clear point of contact for vendors – or for colleagues across the organization. The product owner connects and works across existing business units, embedding with a department (end users) when needed, to do discovery research.[[17]](#footnote-17)

When you are building new software, the product owner is “someone who understands your organization, the problem [you]’re solving, and can advocate for the product [you] ultimately build together. They’ll be responsible for establishing and carrying the long-term vision of the product, implementing a strategy, and guiding its progress, as informed by user research.”[[18]](#footnote-18)

If your organization doesn’t already have a product owner or product manager role, consider creating one (if you are in a position to properly support such a role) or advocating for a (compensated) expansion of your responsibilities to include product ownership.

### Work in the Open and Engage Continuously with Users

Working in the open and centering the end user are key principles and well-accepted best practice in contemporary software. This is especially true in the case of open source software that is developed using agile method (which are, themselves, best practices). To work in the open means “publicly publishing your work on software projects, including existing drafts, future progress, and other work products.”[[19]](#footnote-19) In order to design and implement software well, you need to know your end user and continuously engage them in three key ways:

* **Early user research.** If you are designing new software or writing an RFP to procure new, adapted or existing software, the starting point is the user. In each of these scenarios, the first step in your process will be discovery research with the end user, to identify their needs and pain points. This is also the time to consider the accessibility needs of different user groups and build these in from the start.[[20]](#footnote-20) “User stories” as they’re called, become descriptions of software features. They guide the development of the product and evaluation of the end result.
* **Product evaluation.** Over the course of a development or procurement process, the end user should be involved in evaluating mock-ups and early drafts. This will keep your software close to actual uses and needs. Front-line staff can give feedback on how to best integrate the software with their job, and become empowered to use and maintain the software over time.
* **Ongoing adaptation and maintenance.** Particularly in the case of open source software, the software will be adapted and maintained over time. New features and compliance can be added as standards, policies, and end-user needs change. Users should be empowered to offer suggestions, review potential features, identify bugs, and – if they are competent – to contribute to the code itself.

## Additional Resources

[**DevOps Self-Assessment tool**](https://sara-sabr.github.io/ITStrategy/devops-self-assessment.html): Designed for the Employment and Social Development Canada (ESDC) Information Technology Strategy Team, this questionnaire is a good starting point for other government organizations to assess their team’s technical, process, measurement and cultural capabilities related to software delivery and organizational performance, understand their strengths and challenges, and provide recommendations on specific areas of improvement. It will show the areas where the competencies of your team do not reach the level required to reach team goals.

[**OSPO++**](https://ospoplusplus.com/): OSPO++ is a network and a community of collaborative open source program offices in universities, governments, and civic institutions. It builds resources to help create OSPOs and fosters conversations on how to best manage and grow open source programs.

[**Open Source Policy Examples and Templates**](https://github.com/todogroup/policies)**:** A list of existing open source policies that can be adapted to your organization’s context.

# Module 3: Procurement, Contracting and Budgeting for Open Source Software

## Learning Objectives

In this module we will

* Outline the key differences between licensing and development approaches and why government systems are biased toward proprietary software and waterfall development.
* Discuss different software revenue models and how they affect government cost structures.
* Identify new approaches to procurement, contracting, licensing and staffing.

## Key Points

* **Procurement, contracting and budgeting structures and norms are designed for pre-digital assets – and therefore make it challenging to obtain open source software.** However, as government organizations embrace digital-native approaches, open source software and agile method becomes the easiest path.
* **Open source software requires a different approach to budgeting than proprietary software.** While proprietary software is priced using one of many standard revenue models, open source software is freely distributed, but requires investments in staff capacity and/or service providers to deploy and maintain software.
* **Open source software aligns well with ‘agile’ methods** – which also happen to be the best practice for software development. When procuring custom software services, government should use modular contracting and budgeting approaches, which enable Agile development.
* **Procurement is a creative and exploratory process.** Using procurement correctly, you can not only find the best, lowest cost solution, but also generate positive spin-off effects.
* **Pooled purchasing is an effective strategy** for a peer group of government organizations to share the cost of obtaining and maintaining open source software that fills a common need.

## Section 1: Software Licensing, Revenue and Development Models

### Digital Procurement in Government

Government has well-established norms, processes, and templates for procurement, contracting and budgeting. These were originally designed to obtain non-digital assets and vendor services (like office equipment or snow removal). These assets and services are fixed and predictable. When governments use these traditional processes to obtain software, they tend to favour fixed and predictable software assets (proprietary licenses) and services (waterfall development). They assume that technical expertise is external, and they do not account for peer sharing and communities of practice. But, as one author notes, it’s increasingly the case that “long government budget cycles and mind-sets are a mismatch to the pace of technology and its need for constant improvements and upkeep.”[[21]](#footnote-21)

Public sector executives have an opportunity to embrace a digital-native approach to procurement, budgeting and contracting. These align with software development best practices, while also maintaining the integrity of a legitimate, fair, transparent, and objective public procurement.

Governments are resource-constrained, especially when it comes to adding digital layers onto existing services and business units. Effective procurement processes and intelligent approaches to open source software can help them achieve digital transformation goals within limited budgets.

Digital-native procurement, contracting and budgeting processes empower public sector employees to be in control of how their digital toolsets are designed, managed, and implemented, especially when they are using open source software.

### Software Licensing

Anyone can develop software code and apply whatever license protection they like. A **software license** is a legal document that defines how a piece of software can be used, commercialized, altered, sub-licensed and redistributed. Software licenses are legally enforceable under copyright law.

There are two main categories of software licenses: proprietary and open source.

* A **proprietary license** grants an end user the right to use software, typically for a fee. Users are prohibited from modifying, copying, sub-licensing or distributing the software. Proprietary software is designed to prevent users from accessing the source code.
* An **open source license** grants anyone the rights to use, study, change, and distribute the software and its source code to anyone and for any purpose (often with attribution). The underlying code (“source code”) is available to end users.

The most common examples of open source licenses are [Creative Commons](https://creativecommons.org/), the [GNU General Public License](https://www.gnu.org/licenses/gpl-3.0.en.html), and the [MIT License](https://opensource.org/licenses/MIT). These licenses allow the creator of a work to specify the terms of its use, reproduction and commercialization.

Organizations like the [Free Software Foundation](https://www.fsf.org/) and the [Software Freedom Law Center](https://softwarefreedom.org/) are prominent in the open source community. They are the de-facto enforcers of open source licenses. However, legal action is rare because “open source licenses flow directly from the author/owner to the licensee, [so] the only party in a legal position to enforce [the license protections] is the author or owner of the copyright. Intermediate distributors do not have the power to enforce except with respect to any contribution they may make.”[[22]](#footnote-22)

### Overview of Software Revenue Models

Software development companies have adopted various models for generating revenue from software that is protected under a proprietary license. Here are some of the most common revenue models, along with examples of software that use them.

* **Licensing/one-time purchase.** This entails selling a software product by license that can be used by a single user or a group of users. The general idea is to offer a product that requires making only one payment for it.
  + Microsoft Windows, Apache Server.
* **Subscription (recurring payment).** Unlike licensing, a user receives access to the software by paying a subscription fee on a monthly/annual basis.
  + Netflix, Spotify, Adobe Creative Suite.
* **Pay-per-use.** This pricing tactic is mostly used by different cloud-based products and services that charge you for the computing powers/memory/resources/time used.
  + Amazon Web Services, and Google Cloud Platform.
* **Freemium (upselling).** Freemium is a type of monetization in which a user may access a basic product for free, but will be charged for additional functions, services, bonuses, plugins, or extensions.
  + Skype, Evernote, Gitbook.
* **Hybrid pricing.** Sometimes software companies combine elements of several different revenue models into their pricing plans. For example, a freemium plan might morph into some form of pay-per-use tiered plan. After passing some limit in computation or resources, a user can be forced or offered to use another type of pricing.
  + Mailchimp, Amazon Web Services, SalesForce.[[23]](#footnote-23)
* **Service delivery.** Some software development companies charge for specific services – such as creating a custom instance of an existing Open Source Software. In this case, the client is paying for the time and expertise involved with customizing and implementing software, not the software itself.
  + GNU/Linux, SugarCRM, LibreOffice

### Software Development Models

There are two main software development models.

* **Waterfall** development involves laying out a detailed, linear, sequential plan – including every detail, every feature, every meeting, every benchmark – before starting on a project. The downside of waterfall development is that it does not lend itself well to most software development because there will almost always be a need to adapt to new constraints, requirements, or problems. And since developers aren’t able to foresee every potential problem, it is very likely that the end product will be flawed.
* **Agile** development has an end result in mind, and maybe a timeframe, but it only gets into the details as needed, which allows for change in response to new information. Agile development starts with defining user stories that represent actual use cases and features. Coding happens in “sprints” that last 1-4 weeks, each focused on developing a software feature based on a user story. Using an agile development method, separate but interdependent modules can be developed, tested, deployed and revised easily.

### The Disconnect Between Software Procurement and Software Development

Government procurement is a process designed to get the best value for money and avoid risk. The objective is to identify a need and find the cheapest product that fills that need within a predictable timeframe.[[24]](#footnote-24)

This is a good approach for procuring analog products (e.g., office equipment, streetlights, snow removal service). Defining specifications, comparing options based on responses to an RFP, predicting future costs, and writing a contract for a firm, fixed price (FFP) are all fairly straightforward.

When it comes to software, however, conventional procurement structures will steer you toward purchasing off-the-shelf software or contracting custom software that is developed using the waterfall method based on over-specified, multi-year roadmaps. But that approach has become increasingly misaligned with best practices in the software development industry, which favours agile practices like rapid iteration and continuous user testing to build software that meets user needs. As one expert notes, “this has left a fundamental disconnect between contracting practices (specifying everything is safest) and software development practices (specifying everything is dangerous).”[[25]](#footnote-25)

Conventional procurement – purchasing off-the-shelf software or developing a custom product using the waterfall development model – can *feel like* the least risky option. However, it continually leads to ballooning costs and end products that don’t serve a clear purpose – or worse, end in disaster.

### Matching Best Practice in Procurement and Software

It is important to bear in mind that “when buying custom software, you’re not buying a product. You’re buying a service, the service of developers building software, with features as prioritized by a government product owner.”[[26]](#footnote-26)

Software should be evaluated not only on cost, but also on quality, long-term lifecycle costs of ownership and maintenance, and potential unforeseen concerns like vendor lock-in, increasing license fees, add-on feature costs, maintenance fees, and data ownership.[[27]](#footnote-27)

In the following sections, we will explore what this means for procuring development services to create a custom instance of open source software. We will also explore best practices and tools for procurement and modular budgeting that enables effective agile software development.

## Section 2: Key Issues and Concepts in Digitally-Native Procurement

### Seizing the Opportunity of Procurement

Procurement is traditionally thought of as a compliance procedure that involves detailed a priori specifications, identifies lowest-cost solutions, and prevents corruption. However, procurement is a powerful tool for strategic value-creation.

When administrations seize the opportunities that procurement offers, they can creatively meet broad public sector objectives, promote innovation (inside and outside of government) and drive economic development. In short, they can shape *future public value*.[[28]](#footnote-28)

Governments at all levels are major spenders in their local economies. Significant funding has been earmarked for digital transformation, shared technical services, and IT system upgrades at the federal level.[[29]](#footnote-29)

According to the Business Development Bank of Canada (BDC), there are 41,800 IT businesses in Canada, and 90% of them have 10 employees or less. Together, these businesses contribute $57 billion to the Canadian economy.[[30]](#footnote-30) However, conventional procurement usually favour larger, established vendors, which means that smaller firms and start-ups are systematically shut out of the process.[[31]](#footnote-31) Governments can better support national and local tech communities by engaging with local vendors and writing local sourcing requirements into their RFPs.

### Creating Value with a Procurement Process

At its core, procurement is a process that starts with identifying a fundamental need or opportunity, then exploring and obtaining solutions. If that process is well-designed, it can generate positive spin-off effects (see Box 3.1). Using specific tools like an RFP, the procurement process can specify:

* **Vendor Criteria:** Small, Local, Minority-owned, or Non-profit vendors
* **Solution Criteria:** Public ownership or control, Equitable and sustainable cost models, Functionalities, Outcomes (rather than defaulting to existing solutions)

In the case of open source software, positive spin-off effects might include: fostering a more diverse local ecosystem of technology service providers,[[32]](#footnote-32) sharing software with peer jurisdictions, and maintaining public control over how data is used and commercialized.

#### Box 3.1: Experimenting with Agile Procurement in the Federal Government

Procurement is transforming at the federal level. In 2017, the Treasury Board of Canada Secretariat (TBS) tested a new agile procurement model in conjunction with the Government of Canada’s Open by Default pilot. TBS deployed a “challenge-based procurement process that saw qualified businesses pitch their ideas to a panel of evaluators.”2 Following the presentations, the contract was awarded the same day. In addition, the bidders were able to exchange feedback on the process with TBS procurement staff and evaluators.3

In 2022, Shared Services Canada (SSC) recently launched Agile Procurement Process 3.0 – a new, simplified approach to IT procurement that focuses on procuring a small demo or prototype as a strategy for evaluating potential solutions. This process “lets government officials assess real-world examples of proposed solutions and choose the solution that delivers the best value to Canada and Canadians.”4

1 Alex Benay, “Agile Procurement for Better Digital Solutions,” *Open Government Blog* (blog), July 27, 2017,<http://open.canada.ca/en/blog/agile-procurement-better-digital-solutions>.

2 Treasury Board of Canada Secretariat, “Government of Canada Uses Simplified Procurement to Award Contract for Digital Services,” September 14, 2017,<https://www.canada.ca/en/treasury-board-secretariat/news/2017/09/government_of_canadausessimplifiedprocurementtoawardcontractford.html>.

3 Michael Grant, “An Open, Honest Talk About Procurement Improvements,” Digital Echidna Blog, September 14, 2017,<https://blog.echidna.ca/open-honest-talk-about-procurement-improvements>.

4 Shared Services Canada, “Shared Services Canada Launches Agile Procurement Process 3.0,” March 23, 2022,<https://www.canada.ca/en/shared-services/news/2022/03/shared-services-canada-launches-agile-procurement-process-30.html>.

### Managing the Procurement Process: The Product Owner

Traditionally, procurement is managed by a procurement official, based on detailed budget and functionality specifications. These specifications often come from a variety of sources – such as an IT specialist and an executive focused on policy directives. As a result, specifications can be conflicting, or unclear to the procurement official. More generally, a one-way flow of information (from specialists to procurement officials to vendors) undermines opportunities for creative solution-development.

In a digital-native procurement process, there is a central figure responsible for engaging across business units and advancing the entire procurement process: the Product Owner. The Product Owner does not need to be a technical expert, but should have a working knowledge of software development processes, procurement, and user-centred design methods (see [Module 2](#_heading=h.4w0x83lql6tt) for more detail).

### Discovery: Laying a Strong Foundation by Understanding the Problem

The ‘discovery’ stage is the entry point to software procurement. There are three main elements of discovery:

1. Define the problem to be solved. This involves user research with the beneficiaries and end users of the software (using charettes, brainstorming, etc.).[[33]](#footnote-33)

* Surveys and interviews
* Demos
* User journey mapping
* “Ride alongs”

1. Explore products that already exist. Read how other jurisdictions have approached the same problem. Consult government Github repositories, read blogs and attend conferences.

* Market research and neutral market analysis of existing commercial solutions[[34]](#footnote-34)
* Look at how other jurisdictions have addressed similar problems
* Consult the [GC Open Resource Exchange](https://canada-ca.github.io/ore-ero/en/index.html)

1. Become familiar with vendors. Engage with the world of experts, civic tech enthusiasts and vendors (and let them become familiar with you). Discuss the problems you are trying to solve, and hear their thoughts on existing or potential solutions. Use the following approaches:

* Issue an informal call for ideas (in the civic tech community)
* Issue a formal Request for Information (RFI)
* Host an event related to the problem and invite community groups and vendors[[35]](#footnote-35)
* Reach out to the academic sector

A common misconception is that public officials cannot directly engage with potential vendors. In reality, understanding the market is a crucial starting point for sourcing the best solution. Having a good understanding of what these companies can do is helpful when you are planning to issue an RFP for services. You can also share the needs of your organization so that they become familiar with your long-term roadmap for integrating software in public service delivery. Repeated contracts allow service vendors to become familiar with your technical environment and specific needs. To maintain integrity and avoid concerns, ensure that all vendor engagements are public, well documented, and made publicly accessible as synthesized insights.

### Sourcing Strategies

If you go through a strong discovery process, you will reveal a number of sourcing strategies – buying existing software, using and adapting open source software or contracting a vendor to build net-new software. You should compare these through a fair and objective true lifetime cost of ownership analysis. Based on the results, you will be able to confidently choose the right sourcing strategy for the challenge at hand.[[36]](#footnote-36) There are a number of possible scenarios, each with its own associated costs:

* **If proprietary *and* open source software exists and meets all user needs,** you will choose between procuring existing proprietary software or pursuing an open source software solution.
* **If there is existing open source software that meets all user needs**, an in-house team or hired contractor can deploy a custom instance that integrates with your existing technical environment.
* **If there is existing open source software that needs to be altered** in order to meet all user needs, an in-house team or hired contractor can develop the custom features you need.
* **If there is no existing software that fills the need**, an in-house team or hired contractor can develop software from scratch and release it under an open source license.

A decision tree for choosing a software sourcing strategy.
If proprietary and open source software exists and meets all user needs, you will choose between procuring existing proprietary software or pursuing an open source software solution.
If there is existing open source software that meets all user needs, an in-house team or hired contractor can deploy a custom instance that integrates with your existing technical environment.
If there is existing open source software that needs to be altered in order to meet all user needs, an in-house team or hired contractor can develop the custom features you need.
If there is no existing software that fills the need, an in-house team or hired contractor can develop software from scratch and release it under an open source license.

Figure 3.1: A decision tree for choosing a software sourcing strategy

### Analyzing True Lifetime Cost and Value of Ownership

In order to properly weigh sourcing options, you must perform a True Lifetime Cost and Value of Ownership Analysis, which includes direct and indirect, immediate and long-term costs and benefits.[[37]](#footnote-37) Specifically, this analysis should include:

* The direct cost of the solution (annual license or ownership)
* The technical capacity of staff to build, adapt, integrate and/or maintain software[[38]](#footnote-38)
* The acquisition costs, including all costs of the procurement process and change management (if applicable).
* The operating costs, including annual fees, per-user cost, and insurance.
* The cost of integration with existing digital infrastructure at the city (in terms of staff time and any software or database costs).
* The costs of subsequent releases and disposal, which might include removal of the installation, clean-up costs, releasing code under an open source license, and any associated legal fees.
* The cost of future software customization (work orders) if the city’s needs, processes, or underlying digital infrastructure changes in the future.
* The value of controlling costs, revenue models, updates and additional feature sets (especially for compliance with changing government software performance standards).
* The value of staff capacity-building and empowerment.
* The value of releasing code under an open source license and engaging with a community of peer jurisdictions.

### Evaluating Options

When evaluating existing proprietary software, be critical and cautious of potential downstream contract issues. These might emerge from the vendor’s business model, their approach to ownership or monetization, or software performance and feature updates. This evaluation should influence your choice of strategy.

* **Ownership:** What are the exact licensing conditions? Does the software depend on other proprietary systems (like databases) for which there is an associated cost?
* **Security:** Are there known vulnerabilities? Which party bears responsibility for addressing vulnerabilities?
* **Data governance:** What data is collected? Who owns it? Where is it stored? Who has access to it? How can they use it?
* **Interoperability:** How compatible is it with existing technical infrastructure?
* **Revenue model:** How does the vendor generate its revenue (e.g., fee per “seat;” fee per use; premium features, etc.)?
* **Maintenance:** How regularly is the software updated? What kind of ongoing user support is available?

## Section 3: Contracting and Budgeting for Open Source Software

### Shift Your Focus from Capital Expenditures to Operating Expenditures

Open source software is free, but it takes work to customize and maintain. That means the cost structure will be different. Therefore, budgeting for open source software should account for the shift from *capital expenditure* – buying a product (proprietary software) – to *operating expenditure* – buying a service (development and integration work).

Government organizations will need to hire staff who have capacity to implement and maintain software, or contract technology service providers (writing open contracts for services, as opposed to fixed costs for digital products and licenses).

### Adopt a Modular Contracting Approach for Agile Development

Agile development is structured in short sprints, each focused on building a standalone module. Modular contracting is an approach to financial planning that supports this working cadence.[[39]](#footnote-39)

In a modular contracting scenario, you first establish an umbrella contract for the project through due process and with appropriate approvals. Then you modularize the work and write sub-contracts for specific components, ensuring that the contract amount is a legitimate reflection of the work involved (rather than targeting a number just under the procurement threshold).[[40]](#footnote-40)

In this manner, you will be in control of costs and can ensure that the service provider is meeting deadlines and quality expectations. If they are not, you can break the umbrella contract and award a contract for subsequent to a different vendor – which will be easy to integrate with the work that’s been done already (because it is modular and based on open source components). Modular contracts put you in control, so you can reduce the overall risk (see Box 3.2).

However, be cautious of *bid splitting* or the perception of fraudulent processes. This is when a government “employee splits large contracts into smaller contracts to avoid the scrutiny required for larger contracts.”[[41]](#footnote-41) Although auditing and compliance officials are becoming more and more accustomed to modular contracting for agile development, it is nevertheless important to pre-empt this concern. For example, the US General Services Administration provides “Blanket Purchase Agreements” that help manage and justify groups of modular contracts under a larger contract.[[42]](#footnote-42)

#### Box 3.2: Creating value using modular procurement for California’s new child welfare services system

In 2015, the State of California decided to pursue a different approach to procuring a new Child Welfare Services system, having experienced “a string of failed large IT projects with costs running into the hundreds of millions of dollars.”1

Business-as-usual would have meant issuing “a monolithic multi-year Request for Proposal (RFP) estimated to cost several hundred million dollars and take five to seven years to implement.” 2 Instead, the State of California worked alongside 18F (the technology transformations unit of the federal General Services Administration) and the U.S. Department of Health & Human Services to “use a modular procurement approach coupled with agile design and development methodologies to deliver business value earlier.”3 The project was broken up into smaller modules which could lead to be bid out individually, accelerating the development of the system while reducing overall risk if any one component were to fail.

In addition to the direct benefits to the project itself, several other early wins emerged from the new approach. The project attracted new bidders through its [pre-qualified pool of agile vendors](https://cwscms.osi.ca.gov/CWDSADPQ), and provided an opportunity to ‘work in the open’ and test modules with end users prior to deployment. Additionally, by iteratively refining the RFP process, this project is “providing a template for other projects in other departments.”4

1 Stuart Drown and Mike Wilkening, “Leadership and Innovation at California’s Child Welfare Services,” *18F* (blog), November 17, 2016, <https://18f.gsa.gov/2016/11/17/leadership-innovation-california-child-welfare-services/>.

2 “CWDS Procurement Management Plan,” Child Welfare Digital Services, accessed November 30, 2022, <https://cwds.ca.gov/CWDS_Procurement_Management_Plan>.

3 “CWDS Procurement Management Plan”

4 Stuart Drown and Mike Wilkening, “Leadership and Innovation at California’s Child Welfare Services.”

### Shift from Project-based to Product-based Funding

Your lifetime cost of ownership analysis should have identified ongoing maintenance costs. It is important to create a long-term financial plan for open source software applications. A conventional service contract will focus on the project at hand – e.g. *to build and deliver software*. However, this one-off approach can impede an IT team’s “ability to iterate and improve frequently”[[43]](#footnote-43) especially when it comes to long-term maintenance.

Governments should instead shift towards “planning around product roadmaps, with stable funding for value streams made up of multi-disciplinary team members that can sustain demands and make timely decisions reflecting end-user needs, and a governance that measures progress through working software instead of planning documents.”[[44]](#footnote-44)

In a product-based funding paradigm, a specific business unit (like the Parks Department) can sponsor a project that is carried out by a multidisciplinary Product team, including end users in the Parks Department, IT, and others. The multidisciplinary team identifies the challenge, procures or develops a product, and establishes frameworks for ongoing maintenance.

Staff are therefore responsible for standing up and maintaining a portfolio of products over time. This aligns well with the use of open source software. Using interoperable modules, a well-budgeted and managed team can implement and maintain a large portfolio of software products. Resourcing the IT team must account for direct (labor, software subscriptions and licenses) and indirect (common resources) costing.

The Policy on Service and Digital counsels executives to seek sufficient funding in their program renewal budgets to sustain a product’s full life cycle, including strategically planned improvements. This may mean drawing on different funding envelopes for development and ongoing maintenance. Budgets should account for direct (labour, software subscriptions and licenses) and indirect (common resources) costing. This is known as Product-based Funding because it is focused on the ongoing success of the *product*, not the delivery of a one-off *project*.[[45]](#footnote-45)

### Participate in a Pooled Purchasing Arrangement

Procuring or developing open source software on your own isn’t always feasible, especially for small and mid-sized municipalities. But most jurisdictions need similar software, and each shouldn’t bear the full cost alone. Pooled purchasing is common in other areas (like transportation) but isn’t standard practice for software. Zachary Spicer notes that “shared-services agreements in information technology are much less common than they are in other municipal areas, such as transportation or emergency services, indicating that there is room for municipalities to explore these types of agreements.”[[46]](#footnote-46) This is likely because pooled purchasing is impossible under a proprietary software licensing paradigm; it is prohibited under the license contract. Because open source software is free to use, adapt, and redistribute, governments are fully within their rights to collectively procure development or customization services from a technology service provider and share it amongst themselves. Pooled purchasing is important because:

* Each beneficiary makes a smaller upfront investment
* Better software can be built with mutualized resources
* Tax dollars go further
* Peers contribute to improvements in the shared code[[47]](#footnote-47)

There are several tools and strategies you can use to organize pooled purchasing among a group of organizations (see Module 5).

## Additional Resources

[**A Guide to Reforming Information Technology Procurement in the Government of Canada**](https://govcanadacontracts.ca/it-procurement-guide/): Based on a research project led by Dr. Amanda Clarke in collaboration with Sean Boots, this guide provides a succinct overview of the issues and challenges associated with contemporary tech procurement within the Government of Canada and why it needs to be reformed. To that end, it also advances a set of policy recommendations and emerging best practices for government leaders.

[**Citymart Procurement Institute**](https://medium.com/citymartinsights)**:** A guide for local government officials to better understand how procurement can be part of their governance process, enriching policy and decision-making. This resource contains eight learning units, each covering a logical step in the procurement process. Each unit contains an introduction to the topic, examples, expert insights, cases and worksheets to get you started.

[**Open Contracting Partnership Resources Archive**](https://www.open-contracting.org/resources/): All of the procurement, contracting and budgeting strategies discussed above fall under the general “open contracting” methodology. According to the Open Contracting Partnership, a nonprofit focused on government transparency and efficiency in procurement, “Open contracting is about publishing and using open, accessible and timely information on public contracting to engage citizens and businesses to fix problems and deliver results.” OCP offers a rich library of resources.

[**Interactive Procurement Toolkit for City & Utility Officials**](https://the-atlas.com/procurement-toolkit/): A toolkit to help government officials to develop a clear problem statement and select the most appropriate procurement approach for the issue they want to solve.

[**FVH Application Evaluator: An Open Source Software Tool for Comparing Bids**](https://github.com/ForumViriumHelsinki/ApplicationEvaluator): The Finnish public innovation agency Forum Virium Helsinki has developed open source software that facilitates public procurement. Both a tool and process, FVH Application Evaluator, “is a clear, comprehensive, replicable and easy to adjust for different kinds of procurements. The purpose of the tool is to make evaluation and comparison of proposals easy and agile. It includes graphic coverage which gives a visual overview quickly.”[[48]](#footnote-48) Public administrations can use, replicate and contribute to the code via GitHub.

[**Sample Request for Information (City of Boston)**](https://drive.google.com/file/d/1GJUo6kR9zWmxF4835kHvSq0Rq7QuMLI-/view?usp=sharing): A sample request for information from the City of Boston soliciting submissions in order to gauge interest and gain information about potential solutions for their open data initiative.

[**The Government of Canada Open Resource Exchange:**](https://canada-ca.github.io/ore-ero/en/index.html) A repository of resources for government agencies, including a section for open source software.

[**UnTools:**](https://untools.co/) A collection of thinking tools and frameworks to help you solve problems, make decisions and understand systems.

# Module 4: Open Source Software Security and Maintenance

## Learning Objectives

In this module we will:

* Discuss the interlinked security and maintenance considerations associated with both conventional proprietary and open source software.
* Outline key challenges that organizations face in maintaining and securing their open source software solutions
* Explore strategies for ensuring successful stewardship and maintenance of open source software.

## Key Points

* **There is no such thing as zero-risk software: all software – proprietary and open source – is built on sub-modules and involves risks that must be managed.** In fact, most conventional proprietary companies use open source code in their products and participate in maintaining open source software.
* **Open source software is not inherently less secure than conventional proprietary software**. As with any software, open source software carries certain risks that are unavoidable, but this should not discourage you from considering it for your organization. The best course of action is to conduct a thorough risk assessment and make appropriate plans to mitigate those risks.
* **More ‘eyes’ on a piece of open source software increase the chances that critical maintenance issues and security vulnerabilities will be resolved.** Open source software that is backed by a large active community of users and developers is more likely to receive regular updates and security patches. However, if you are using open source software that is not widely supported by a community of users, you will need to compensate by making sure you have appropriate in-house capacity (or can hire external support) to monitor for security vulnerabilities.
* **Don’t fork it unless you can maintain it; the more bespoke code your organization creates, the more the maintenance burden falls on you**. If you create an independent local instance (forking the code) updates and security patches are your own responsibility – and they become increasingly difficult, as your version and the main branch grow further apart over time.

## Section 1: Introduction to Open Source Software Security and Maintenance

### What are Open Source Software Security and Maintenance?

As one software security expert puts it, “unlike wine and cheese, software does not get better with age – in fact, its security strength decreases over time. This is because of software obsolescence.”[[49]](#footnote-49) When it comes to open source software, security and maintenance go hand-in-hand.

* **Open source software security** is about “maintain[ing] the confidentiality, integrity, and availability of information resources in order to enable successful business operations.”[[50]](#footnote-50)
* **Open source software maintenance** is about keeping software functioning properly and preventing it from becoming obsolete.

Maintenance is critical, and failing to invest adequately in it carries “hidden costs”: *direct* costs associated with exploitation of security vulnerabilities and subsequent loss of functionality and *indirect* costs resulting from (among other things) the “loss of qualified labor and slower growth and innovation.”[[51]](#footnote-51)

### Proprietary vs. Open Source Software Security and Maintenance

There is a common misconception that open source software is less secure than conventional proprietary software. With proprietary software, maintenance is the responsibility of the vendor. Additionally, with conventional proprietary software, service agreements also usually include assurances in the form of liability and warranty clauses. This contributes to the assumption that it is more secure.

By contrast, with open source software, the responsibility for maintenance is more diffuse and ultimately relies on the community of users. However, this fact does not mean that open source software is less secure. One report found that 89% of large IT firm executives believed open source software to be as secure as proprietary software.[[52]](#footnote-52) In short, *“licensing models have nothing to do with security.”*[[53]](#footnote-53)

### Software and Privacy in the Digital Age

Governments, by their very nature, generate and use large amounts of personally-identifiable information about their residents in order to administer programs and services and make informed policy decisions.

Virtually all personally-identifiable information nowadays is in digital form, which makes it much more efficient to manage through its lifecycle using an array of software applications, but also makes it more vulnerable to attack. Even non-personally identifiable information can be pieced together to build up an accurate profile of an individual – including their location, habits and preferences.[[54]](#footnote-54)

Privacy is considered a foundational component of freedom in our society.[[55]](#footnote-55) Personally identifiable information stored in an improperly secured location – whether it is on a local server or in the cloud – is vulnerable to privacy breaches. For this reason, safeguarding information – including personally identifiable information – must be a key priority when considering the security of both conventional proprietary and open source software.

## Section 2: Key Issues and Concepts in Maintaining and Securing Open Source Software

### Open Source Software Security Risks

There are several aspects of open source software which can render it vulnerable to security risks including:

* **Excessive access:** Since code is open for all to see, bad actors can take advantage of this vulnerability to manipulate code in potentially harmful ways.[[56]](#footnote-56)
* **Lack of verification:** The nature of how open source software is developed means that quality assurance and security testing are not guaranteed.[[57]](#footnote-57)
* **Lack of support:** Open source software that is backed by a large active community of users and developers is more likely to receive regular updates and security patches. However, if you are using open source software that is not widely supported by a community of users, you will need to compensate by making sure you have appropriate in-house capacity (or can hire external support) to monitor for security vulnerabilities.

### The Limitations of the Open Source Community Support Model

Most active open source communities regularly update software and issue patches for known vulnerabilities. However, issues can arise if there is no one to update the software either because the software too new or too nice to attract a large enough developer community or the existing community has not been maintaining an open source project.

A cybersecurity attack that results in a leak of data containing sensitive information can have a seriously negative impact on trust in a government organization. In such a case, waiting for updates and security patches to be developed by the open source project community may not be an option. Instead, government organizations need to be able to quickly assess and solve the problem which means having some internal expertise in the open source modules or software used by the organization.[[58]](#footnote-58)

### Conventional Risk Management Approaches

Most, if not all, government organizations will already have an IT security framework in place. An organizational IT security policy establishes an organization's general security objectives. It will outline how employees work and collaborate on corporate networks while respecting the organization’s position on managing security risks.

In many cases, the measures in place to mitigate security risks stemming from conventional proprietary software will also cover open source software.[[59]](#footnote-59) In fact, current practices to approach risk assessment can be applied to secure open source software in addressing vulnerabilities and minimizing their impact.

Your organization more than likely has an established approach to IT risk management and will probably have a plan in place. Even if there are multiple versions of how to approach risk management, once you have identified a risk there are four basic ways to handle it: avoid, mitigate, transfer, or accept.

The problem with conventional proprietary software is that identifying risks in the first place can be challenging because they may be hidden behind the vendor’s sales pitch and fine print in the license contract.

### Managing Risks for Open Source Software

Open source risk management can be approached in a similar way. A recent framework released by Google identifies three largely independent problem areas with a concrete objective: knowing about vulnerabilities in your open source software, preventing the addition of new vulnerabilities, and fixing (or removing) vulnerabilities.[[60]](#footnote-60)

Knowing your vulnerabilities involves your team being able to capture precise vulnerability metadata from all available data sources, preferably following industry standards to simplify tracking tasks. At the moment, better tooling to understand quickly newly discovered vulnerabilities is needed. However, understanding the inherent risks of new dependencies is a great start. Fixing vulnerabilities is not always an easy task, as directly solving vulnerabilities will require some in-house expertise that your team may lack. However, if your organization needs to address a specific vulnerability, the best course of action is to work bottom-up through the dependency chain for the software to be fixed.

As new threats emerge, new vulnerabilities will become apparent, which means that managing risk is an ongoing activity that needs to be constantly updated, as existing controls may become ineffective. One way to keep track of risks and an organization's risk appetite is by using a risk grid that can help organizations evaluate their likelihood of risks and understand and model the relationships among threats, vulnerabilities and controls that need to be applied.[[61]](#footnote-61)

Ultimately, open source vulnerabilities should be approached as an extension of an organization’s general IT security objectives and reporting processes rather than as an entirely new task. Remember that understanding your risks will help create a risk-aware culture within the organization while considering how technology risks – including those of open source software – align with business objectives.

### Software Dependencies: A Challenge for Security and Maintenance

All software is built on other software to one degree or another. Developers can build new applications much faster by pointing to code that has already been developed by a third party than they could if they were writing new code from scratch. In fact, a 2017 investigation found that, out of over 1100 commercial codebases, “96 percent of the scanned applications contain[ed] open source components, with an average 257 components per application.”[[62]](#footnote-62)

When ‘Program A’ requires ‘Program B’ to function, this is called a **software dependency**. Dependencies can be direct (i.e., Program A directly depends on Program B in order to run) or transitive (i.e., Program A depends on Program B which in turn depends on Program C for its functioning).[[63]](#footnote-63) Dependencies are largely unavoidable, but those that are outside your control present a risk.

A dependency failure can occur when a code library or package critical to the functioning of a given piece of software is altered or deleted. The 2016 ‘left-pad’ incident is an example that illuminates how significant the impacts of a dependency failure can be:

* In 2016, a code package known as “[left-pad](https://www.npmjs.com/package/left-pad)''[[64]](#footnote-64) which consisted of just 11 lines of code was removed by its developer from the NPM Registry where it had been hosted. Its removal caused a cascade of dependency failures, resulting in software crashes worldwide.[[65]](#footnote-65)
* As one blogger put it, the larger issue illustrated by this dependency failure is that “so many packages and projects took on a dependency for a simple left padding string function, rather than their developers taking 2 minutes to write such a basic function themselves.”[[66]](#footnote-66)

The bottom line: the more dependencies a piece of software has that are outside your control, the greater the security risk and the greater the odds that something will break. The first step toward managing dependencies and mitigating downstream impacts on security and maintenance of your open source software is simply to be aware of them. To support this, many software developers are now including a Software Bill of Materials (SBOM) that lists all open source and third-party components present in a codebase.[[67]](#footnote-67)

## Section 3: Strategies for Maintaining and Securing Open Source Software

### Leverage Your Organization’s Existing Data Governance Framework

Government organizations should have a thorough understanding what data they are collecting, using, and storing and how it supports them in achieving their objectives. Having an organization-wide data governance framework in place formalizes this understanding and provides a reference point for all data activities whether they involve proprietary or open source software.

Specific data governance considerations for open source software include policies and procedures for ensuring that personally identifiable information is kept out of open source codebases,[[68]](#footnote-68) as well as “keep[ing] sensitive data such as credentials secure and separate from source code” and “avoid[ing] storing keys and other sensitive material in systems not approved for that purpose.”[[69]](#footnote-69)

### Make Privacy and Security a Priority

Public sector executives should foster a culture of data protection and incentivize their teams to adopt privacy and security by design practices[[70]](#footnote-70) to make sure all software – proprietary and open source – is free from vulnerabilities that could lead to a privacy breaches or loss of functionality.[[71]](#footnote-71)  You can do this by:

* Regularly reviewing policies to ensure they align with the current threat environment
* Organizing regular training sessions for staff on IT security topics

These practices should be reviewed and updated regularly so that they reflect the current threat environment. And while not specific to open source software, it is important to have a plan for ensuring business continuity and assessing how the organization will react in case of a security breach or loss of data for unintended reasons (such as the loss of a computer).

### Establish Policies & Procedures for Regular Code Review and Vulnerability Disclosure

If you have developers on your team supporting open source applications used by your organization, it’s important that you establish appropriate policies and procedures for reviewing source code for potential security risks. When code is regularly reviewed, it “increase[s] the likelihood of catching bugs, security vulnerabilities, and reduces the risk of committing sensitive data.”[[72]](#footnote-72) Code reviews also allow you to assess the level of dependency in your open source software and minimize exposure to dependency failures.

There are a number of tools and services that can help your team quickly evaluate open source software for known security vulnerabilities and potential dependency issues, including the [Open SSF Scorecard](https://securityscorecards.dev/) tool which “checks for vulnerabilities affecting different parts of the software supply chain including source code, build, dependencies, testing, and project maintenance.”[[73]](#footnote-73)

### Don’t Fork Code Unless You Can Maintain It

Maintaining and securing any type of software – proprietary or open source – means managing a large number of open source components and dependencies to mitigate security risks. That’s easier when your software is an active part of the open source community.

However, if your organization creates an independent local instance (forking the code) updates and security patches are your own responsibility – and they become increasingly difficult, as your version and the main branch grow further apart over time. The bottom line: Don’t fork it unless you can maintain it.

Government of Canada guidelines for open source software recommend that, “wherever possible, use open source software without modification or contribute back to the project… Use configuration and customize the software with modules, plugins or extensions and make those available to the community.” [[74]](#footnote-74)

## Additional Resources

[**2022 Open Source Security and Analysis Report**](https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html): This report by Synopsys “examines vulnerabilities and license conflicts found in more than 2,400 codebases across 17 industries. The report offers recommendations to help security, legal, risk, and development teams better understand the security and risk landscape accompanying open source development and use.”

[**OpenSSF Scorecard**](https://securityscorecards.dev/): This tool checks for vulnerabilities affecting different parts of the software supply chain including source code, build, dependencies, testing, and project maintenance. Each automated check returns a score out of 10 and a risk level. The risk level adds a weighting to the score, and this weighting is compiled into a single, aggregate score. This score helps give a sense of the overall security posture of a project. Alongside the scores, the tool provides remediation prompts to help you fix problems and strengthen your development practices.

[**Concise Guide for Evaluating Open Source Software**](https://github.com/ossf/wg-best-practices-os-developers/blob/main/docs/Concise-Guide-for-Evaluating-Open-Source-Software.md#readme): This guide, aimed at software developers, provides a series of technical questions to work through before adopting open source software dependencies or tools to evaluate their security and sustainability.

# Module 5: Collaborating on Open Source Software

## Learning Objectives

In this module we will:

* Review the interjurisdictional issues raised in previous modules and key considerations when forming multi-jurisdictional teams around a particular piece of software, including resource-sharing between different jurisdictions or levels of government
* Explore what it means to participate in a global community of codebase stewardship
* Demonstrate the value of releasing your software under an open source license

## Key Points

* **Collaboration on open source software is a win-win.** The software you build, or the adaptations you make, will be useful to other jurisdictions as well. Open source software performs better and becomes more secure as the community around it grows. It is therefore in each user’s best interest to contribute, and to add more contributing users.
* **Government organizations do not need to reinvent the wheel.** When source code is released under an open source license, any jurisdiction can make small software adaptations to ensure that the software is best-fit to local use cases, regulations and standards. When a core set of functionalities are shared, each jurisdiction’s time, effort and resources can be focused on making small adaptations to fit the software to their local context (rather than building a full set of core functionalities from scratch).
* **Same-level governments are natural collaborators when it comes to designing, obtaining and maintaining open source software.** They are responsible for delivering a similar set of services and have a similar set of needs (internal, administrative and external, resident-facing). They are therefore natural collaborators when it comes to designing, obtaining and maintaining open source software.
* **Jurisdictions already collaborate in various well-established networks and peer groups that can be activated as open source software stewardship communities.** Distributed version control platforms (like GitHub) enable users to contribute or suggest changes in a structured and collaborative way.

## Section 1: What Does It Mean to Work with Open Source Software across Jurisdictions?

### Benefits of Collaboration

**Collaboration on open source software is a win-win.** The software you build, or the adaptations you make, will be useful to other jurisdictions as well. Open source software performs better and becomes more secure as the community around it grows. Each user benefits from continuous improvements to software (in the form of updates, new features or security improvements). It is therefore in each user’s best interest to contribute, and to add more contributing users.

For government organizations, another benefit of collaboration is that they do not need to constantly reinvent the wheel. When source code is released under an open source license, any jurisdiction can make small software adaptations to ensure that the software is best-fit to local use cases, regulations and standards. When a core set of functionalities are shared, each jurisdiction’s time, effort and resources can be focused on making small adaptations to fit the software to their local context (rather than building a full set of core functionalities from scratch). Knowledge and resource sharing help partners with less experience in open source software to build capacity.

### Who Can You Collaborate With?

Governments and public sector agencies that are interested in using open source software should join existing networks or build collaborative relationships with peer jurisdictions. These networks might surround a specific software, or they might be focused more broadly on using open source software in the public sector. Within such communities, users can share challenges, opportunities, capacities, resources, and processes (see Box 5.1).

Governments at the same level – whether municipal or provincial -- are natural collaborators when it comes to designing, obtaining and maintaining open source software. They are responsible for delivering a similar set of services and have a similar set of needs (internal, administrative and external, resident-facing). And when procuring new software, a pooled purchasing approach allows a group of similar government agencies to take advantage of economies of scale.

#### Box 5.1: CKAN open data management platform

An excellent example of open source collaboration across jurisdictions is the CKAN open data management platform. Developed by the non-profit Open Knowledge Foundation (OKFN) for the Government of the United Kingdom, National and regional governments around the world use CKAN as the foundation of their open data portals, including the [Government of Canada](https://open.canada.ca/en/open-data) the [Government of British Columbia](https://www2.gov.bc.ca/gov/content/data/bc-data-catalogue), and the [Government of Ontario](https://data.ontario.ca/about).

Since code improvements are shared, each of the [more than 115 governments and institutions](https://ckan.org/government) benefits from producing a better product at a reduced cost to individual consumers. In addition to helping governments develop and implement instances of the platform customized to their needs, CKAN’s sizable user and developer community continually generates and disseminates knowledge via GitHub, mailing lists, IRC channels.

1 British Columbia. Ministry of Jobs, Economic Recovery and Innovation, “About the Catalogue Open Source Project,” accessed November 3, 2022,<https://www2.gov.bc.ca/gov/content/data/bc-data-catalogue/about-catalogue-open-source-project>.

### Barriers to Working with Open Source Software across Jurisdictions

Each of the areas covered in the previous modules can present specific challenges when it comes to collaborating on open source software projects across jurisdictions or between public agencies.

* **Organizational and staff capacity challenges:** There can be significant variation in digital capacity between governments and public institutions, especially when you get down to the municipal level.[[75]](#footnote-75) Not all governments will have open source teams or product manager roles; smaller municipalities may only have one dedicated person responsible for IT. When it comes to governing open source projects, political – not technical – skills are required. Software developers, users, and maintainers may not be the ones best suited to manage issues like accountability and negotiation of shared costs – and by the same token, product owners may not understand the dynamics and implications of technical aspects of software maintenance.
* **Procurement, contracting and budgeting challenges:** Standard procurement processes, templates and norms prioritize commercially available products from established vendors – and in the case of technical service delivery – prioritize waterfall-style software development. Governments may be locked into long-term contracts with vendors, which prevents them from being in a position to consider collaborations around open source alternatives. Furthermore, some organizations have a “not invented here” mentality when it comes to software, and neglect to look at what other government organizations have done. A common mantra in the open source software community is that the wheel has already been invented. This applies to full software as well as sub-modules. Governments can, and should, discover what has worked elsewhere.
* **Security and maintenance challenges:** Open source software projects undertaken by a group of government actors, especially for relatively niche use cases, may become neglected and obsolete unless a governance structure is established that sets out responsibilities for ongoing maintenance, and there is funding for maintenance service provision. Additionally, differences in digital policy between jurisdictions (e.g. user consent and data protection requirements under GDPR in the EU versus Canada’s policy) may create obstacles to developing open source software that meets the risk management needs of all parties in an international collaboration.

### Merging Forked Code

Merging forked code back into the main branch is a challenging moment in an open source software collaboration. Best practice, in open source software development, is to make small incremental changes, thereby minimizing the risk of breaking dependencies. When everyone working on the code is in sync, any changes affect each other with minimal disturbance. If someone creates a fork and makes many changes, then eventually wants to merge the codebases back together, there may be conflicts – both structural and functional.

#### Box 5.2: The *Notify* messaging service

*Notify* is an open source software tool that participating government agencies can use to easily and securely send out thousands of personalized emails, text messages, and postal letters to their contacts.1

The United Kingdom’s Government Digital Service first released their Notify platform as open source, which allowed other governments to adopt it for their own purposes. For example, the Government of Canada developed its own version called [GC Notify](https://notification.canada.ca/) by modifying it to support multiple languages.2

1 Beeck Center for Social Impact and Innovation, “Notify,” Intergovernmental Software Collaborative, accessed October 18, 2022, <https://softwarecollaborative.org/cooperatives/notify.html>.

2 Bryan Willey, “Introducing Notify,” *Canadian Digital Service* (blog), November 26, 2019,<https://digital.canada.ca/2019/11/26/introducing-notify/>.

Consider an open source application such as Notify (see Box 5.2) as a hypothetical example: the original application may have been coded to disallow messages from being edited. A forker may subsequently build a trust feature based upon the fact that messages cannot be changed. However, in the meantime the main branch code has been changed to allow editable messages. If the forker decides to merge back, they will find that functionality in the original codebase has changed, and each branch has built user-facing features based on different premises. This creates a dilemma: *Should the merged codebase allow message editing (which would undermine one branch’s trust features) or revert to non-editable messages (which would undermine any tools and workflow involving editable messages)?*

To avoid situations like the one described above, the Government of Canada guidelines for open source software recommend that staff “wherever possible, use open source software without modification or contribute back to the project” and “use configuration and customize the software with modules, plugins or extensions and make those available to the community.”[[76]](#footnote-76)

### Releasing Code under an Open Source Licence

The Government of Canada recommends that government agencies who have developed their own software should consider releasing the code under an open source license, and provides guidelines on doing so effectively. “It is recommended that where they have the right to do so, departments publish all source code as open source software, whether the software solution was (i) acquired as OSS; (ii) developed in-house by GC employees or (iii) acquired through the terms of procurement contracts where appropriate license terms were negotiated.”[[77]](#footnote-77)

There are numerous legal considerations when selecting a license for your organization’s open source software project.[[78]](#footnote-78) Some governments (e.g., [British Columbia](https://github.com/bcgov/BC-Policy-Framework-For-GitHub/blob/master/BC-Open-Source-Development-Employee-Guide/Licenses.md)) have published guidelines for choosing an appropriate license that are specific to their legislative context. For example, if your open source project is an adaptation or derivative of an existing open source project, the best practice is to use the same license as the original; if it is a new open source project, the license you select will depend on the desired outcome and the licensing of any third-party software used in your project.[[79]](#footnote-79)

## Section 2: Strategies for Collaborating Across Jurisdictions on Open Source Software

Despite the potential challenges, it is possible to create productive open source software collaborations interjurisdictionally.

### Build a Governance Structure Intentionally

Collaborations that start without a clear governance structure may struggle, especially if too many partners are involved. It may be difficult to agree on what collaboration model is being followed and determine appropriate roles for each partner.[[80]](#footnote-80)

When considering a collaboration, take the time to align with potential partners. Identifying a shared need or purpose for an open source application is one of the first steps to beginning a collaboration. By starting conversations with your counterparts in other government or public sector agencies, you will gain a better understanding of what their needs are, where there may be potential points of alignment, and what shape your collaboration should take.[[81]](#footnote-81)

Once that initial contact has been established, partners can get into the specifics such as “planning and documenting the proposed governance structure, principles of working in the open, and their alignment on how to resource their shared work.”[[82]](#footnote-82) One way to approach these conversations is to use a tool like the *Governance Game* developed by the Foundation for Public Code: The *Governance Game* “an interactive game exploring governance of a public codebase. It helps participants reflect on what governances means for a codebase, the complexity around it, and surfaces issues worth considering during set up.”[[83]](#footnote-83)

### Create Communities of Practice around Open Source Software

Open source champions across government agencies can form communities of practice around open source software. Tapping into pre-existing peer group networks of government staff responsible for delivering similar programs or services can be an effective way of developing and sharing open source software specific to certain areas of focus or use cases (e.g., transportation planning, asset management). CoPs with a general open source focus can be useful for exchanging knowledge on open source at an organizational level. CoPs can steward shared repositories of open source software modules and resources.

### Create or Participate in a Pooled Purchasing Arrangement

Procuring or developing open source software on your own isn’t always feasible, especially for small and mid-sized municipalities. With pooled procurement (also known as cooperative purchasing) you don’t need to bear the cost alone.[[84]](#footnote-84) Pooled purchasing increases the efficiency of a procurement process and reduces costs for participating organizations. The advantages of pooled purchasing for government include:

* Lower prices
* Quality controls
* Shortened delivery lead times
* Streamlined processes
* Access to alternative financing
* Technical cooperation
* Increased transparency and information sharing

Advantages also extend to suppliers:

* Repurposing existing / shared code
* Reduced transaction costs
* Service demand predictability (feature development and maintenance)
* Upfront financing for working capital and demand forecasting[[85]](#footnote-85)

Creating or Joining a Cooperative Procurement

There are two main approaches to pooled purchasing. Joint solicitation (coordinating a shared process up-front) or “piggybacking” (sharing work that’s already been done across public agencies).[[86]](#footnote-86) These are the main steps to follow for joint solicitation:

1. The cooperative is formed when one or more parties identify a common requirement suitable for cooperative purchase and sign a written agreement to cooperate.
2. Lead party solicits proposals and awards contract(s).
3. Contract is available for use.
4. Participating entities sign an agreement (NASPO/WSCA calls it a “participating addendum”) in the specific contract(s). This is necessary to get users’ statutory requirements included as well as for the lead entity to administer efficiently.[[87]](#footnote-87)

Piggybacking is a more ad-hoc approach. In the U.S., there are tools such as [CoProcure](https://www.coprocure.us/) that can help you find and join geographically and topically relevant products and solicitations. However, there is currently no ‘one-stop shop’ for cooperative purchasing opportunities in Canada; opportunities to piggyback on other organizations’ contracts will likely come through your connections to existing peer networks.

### Properly Document Open Source Projects

You can invest significant time and effort into developing a piece of open source software, but if no one else can understand what the software is and does, on a high level, and how the modules work, on a technical level, then the chances of its being adopted by others outside your team are low. To make it easier for others to adopt open source software you’ve developed, it’s essential to spend time properly documenting it.

Consider using the Standard for Public Code, “a set of criteria that supports public organizations in developing and maintaining software and policy together.”[[88]](#footnote-88) The Standard serves as a framework for governments and public agencies to develop their own open source projects in a way that is transparent, accountable, and understandable by others.

## Additional Resources

[**Introduction to the Standard for Public Code**](https://citybycity.academy/course/standards-for-smart-cities) (CITYxCITY Academy): This free course introduces the term public code, explains why this is important, and introduces the process through which software and policy code can become certified public code.

[**Intergovernmental Software Collaborative (Beeck Center for Social Impact and Innovation)**](https://softwarecollaborative.org/): This community of practitioners from all levels of government across to U.S. shares information, case studies, and best practices for developing intergovernmental software collaboratives.

**[Open Source Guides: Leadership and Governance (GitHub)](https://opensource.guide/leadership-and-governance/)**: A general resource providing guidance on establishing and formalizing roles and governance structures as your open source project grows.

**[Co-procure](https://www.coprocure.us/)**: A U.S.-based platform for finding current government software procurements or starting a new collaborative procurement.

# Glossary

|  |  |
| --- | --- |
| Agile software development | “Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.” Agile software development involves a number of process management strategies, like daily stand-ups, scrums, user journeys and [other basic tools](https://agile.18f.gov/agile-fundamentals/). (Source: [Atlassian](https://www.atlassian.com/agile). See also the [Agile Manifesto](http://agilemanifesto.org/)) |
| Bid Splitting | When a government “employee splits large contracts into smaller contracts to avoid the scrutiny required for larger contracts.” (Source: [The Association of Certified Fraud Examiners](https://www.fraudconference.com/uploadedFiles/Fraud_Conference/Content/Course-Materials/presentations/23rd/ppt/post-06-Fraud-in-Procurement-Without-Competition.pdf)) |
| Branch repository | See: Repository |
| Central repository | See: Repository |
| Commercial off-the-shelf (COTS) | “Commercial off-the-shelf or commercially available off-the-shelf (COTS) products are packaged or canned (ready-made) hardware or software, which are adapted aftermarket to the needs of the purchasing organization, rather than the commissioning of custom-made, or bespoke, solutions.” Source: [Commercial off-the-shelf - Wikipedia](https://en.wikipedia.org/wiki/Commercial_off-the-shelf) |
| DevOps | “DevOps can be best explained as people working together to conceive, build and deliver secure software at top speed. DevOps practices enable software developers (devs) and operations (ops) teams to accelerate delivery through automation, collaboration, fast feedback, and iterative improvement. Stemming from an Agile approach to software development, a DevOps delivery process expands on the cross-functional approach of building and shipping applications in a faster and more iterative manner.” (Source: [GitLab](https://about.gitlab.com/topics/devops/)) |
| Distributed version control | “In software development, distributed version control (also known as distributed revision control) is a form of version control in which the complete codebase, including its full history, is mirrored on every developer's computer. Compared to centralized version control, this enables automatic management branching and merging, speeds up most operations (except pushing and pulling), improves the ability to work offline, and does not rely on a single location for backups. Git, the world's most popular version control system, is a distributed version control system.” Source: [Distributed version control - Wikipedia](https://en.wikipedia.org/wiki/Distributed_version_control) |
| Forking | “A fork is a copy of a repository that you manage. Forks let you make changes to a project without affecting the original repository. You can fetch updates from or submit changes to the original repository with pull requests.” [Source: GitHub](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/about-forks) |
| Free (aka libre) software | “Free software or libre software, infrequently known as freedom-respecting software, is computer software distributed under terms that allow users to run the software for any purpose as well as to study, change, and distribute it and any adapted versions. Free software is a matter of liberty, not price; all users are legally free to do what they want with their copies of a free software (including profiting from them) regardless of how much is paid to obtain the program” Source: [Free software - Wikipedia](https://en.wikipedia.org/wiki/Free_software) |
| Free and Open Source Software (FOSS) | “Free and open-source software (FOSS) is software that is both free software and open-source software.” Source: [Free and open-source software - Wikipedia](https://en.wikipedia.org/wiki/Free_and_open-source_software) |
| Merge request | See: Pull request |
| Modular Budgeting | Modular budgeting is financial planning that supports modular contracting (see above). A modular budget applies to the person-hours and other costs associated with a discrete module. |
| Modular Contracting | “Modular contracting is an acquisition strategy that… break[s] up large, complex projects into multiple, tightly-scoped procurements to implement technology systems in successive, interoperable increments.” (Source: [18F](https://18f.gsa.gov/2019/04/09/why-we-love-modular-contracting/)) |
| Open source software | “Open source software is software with source code that anyone can inspect, modify, and enhance. ‘Source code’ is the part of software that most computer users don't ever see; it's the code computer programmers can manipulate to change how a piece of software – a ‘program’ or ‘application’ – works. Programmers who have access to a computer program's source code can improve that program by adding features to it or fixing parts that don't always work correctly.” (Source: [OpenSource](https://opensource.com/resources/what-open-source)) |
| Problem Statement / Problem Framing | “Problem framing is a problem-solving method that’s designed to align the entire team with one solution for a project by structuring the issue’s details in a digestible and collaborative way. So, when your team can’t agree on a solution, use this play to take a step back and align on the problem you are solving for.” (Source: [Atlassian](https://www.atlassian.com/team-playbook/plays/problem-framing)) |
| Product Owner | “Someone who understands your organization, the problem we’re solving, and can advocate for the product we ultimately build together. They’ll be responsible for establishing and carrying the long-term vision of the product, implementing a strategy, and guiding its progress, as informed by user research.” (Source: [18F](https://18f.gsa.gov/partnership-principles/)) |
| Pull request | “Contributions to a source code repository that uses a distributed version control system are commonly made by means of a **pull request**, also known as a **merge request**. The contributor requests that the project maintainer pulls the source code change, hence the name "pull request". The maintainer has to merge the pull request if the contribution should become part of the source base.” Source: [Distributed version control - Wikipedia](https://en.wikipedia.org/wiki/Distributed_version_control#Pull_requests)  See also: Distributed version control |
| Repository | “In version control systems, a **repository** is a data structure that stores metadata for a set of files or directory structure.” Source: [Repository (version control) - Wikipedia](https://en.wikipedia.org/wiki/Repository_(version_control))  In a distributed version control system “every project has a **central repository** that is considered as the official repository, which is managed by the project maintainers. Developers clone this repository to create identical local copies of the code base. Source code changes in the central repository are periodically synchronized with the local repository.  The developer creates a new branch in their local repository and modifies source code on that **branch**. Once the development is done, the change needs to be integrated into the central repository.”  See also: Distributed version control |
| Request for Information (RFI) / Sources Sought Notice (SSN) | **Request for Information (RFI) / Sources Sought Notice (SSN):** A market research tool used to obtain price, delivery, capabilities, interest, etc. for planning purposes.” (Source: [US Government Services Administration](https://www.gsa.gov/small-business/small-business-resources/training-resources/rfp-rfi-and-rfq-what-are-the-differences)) |
| Request for Proposals (RFP | “A request for proposal (RFP) is both the process and documentation used in soliciting bids for potential business or IT solutions required by an enterprise or government agency. The RFP document typically outlines a statement of requirements (SOR) to be met by prospective respondents wishing to make a bid to deliver the required solutions. It might cover products and/or services to meet the given requirements. The RFP documentation also typically covers the related procurement process, evaluation criteria, commercial terms and conditions, timeliness and activities involved, and what respondents should include in their RFP response.” (Source: [Gartner](https://www.gartner.com/en/information-technology/glossary/request-proposal-rfp)) |
| Request for Quote | **“**A solicitation method used to obtain price, cost, delivery, and related information from suppliers.” (Source: [US Government Services Administration](https://www.gsa.gov/small-business/small-business-resources/training-resources/rfp-rfi-and-rfq-what-are-the-differences)) |
| Sole-source | “Procurement by noncompetitive proposals, referred to as sole source procurement, is procurement through solicitation of a proposal from only one source. Sole source procurements must adhere to standards,” that govern how a government body can justify that only one vendor is able to provide the requested services. (Source: [US Department of Justice](https://www.ojp.gov/sites/g/files/xyckuh241/files/media/document/Sole_Source_FactSheet_C.pdf)) |
| Total Cost Analysis / True Cost Analysis | This includes the full cost of building or procuring software, implementing it and maintaining it. TCA includes direct and indirect, immediate and long-term costs (such as staff time, license fees, etc.). |

# Appendix A: Executive One-Pagers

## 

# Executive One-Pager #1: Why Open Source Software?

**Public sector executives and senior managers have an opportunity to lead digital transformation of mission-critical service delivery and operations within their organizations by embracing open source software solutions.**

**To drive this transformation, you need to:**

* Re-orient your organizational culture to become open source software friendly
* Invest in building up your staff capacity
* Champion digitally-native approaches to procurement, budgeting and contracting that support open source software adoption
* Understand the fundamental differences between conventional proprietary software and open source software
* Leverage the potential of cross-jurisdictional collaboration to build and support open source software applications

## Key Points

* **Governments and public sector agencies across Canada and worldwide are already using open source software in significant ways.** As governments at all levels incorporate software into a wide variety of processes and services to meet their objectives – from efficient public service delivery to social and environmental goals – open source software presents an excellent option for ensuring quality, ongoing control, and resource efficiency.
* **Open source software is fundamentally different from conventional proprietary software** in several ways: how it is licensed, owned and managed; how it is developed and maintained; and how it is financed and commercialized. Open source software is freely distributed under an open-source license. Many open source software projects are maintained by a global community of users and developers, while others have a smaller contributor base. While the software itself is free, your organization may invest in in-house staff capacity to integrate and maintain software, or contract with a vendor to add specific features or do a custom integration with your existing digital environment.
* **Open source software does not have to be built from scratch.** Hundreds of full open source software projects and even more small component open source modules offer as many options and functionalities as conventional proprietary software. These open source projects and modules serve as a foundation upon which new open source software can be built.
* **Open source software has the potential to bring a variety of unique benefits to government** – including customizability, interoperability, and transparency. While there are barriers to adopting open source software in government – including a lack of in-house technical capacity and inflexible procurement practices – fortunately, there are a variety of strategies and best practices that can help you to overcome these barriers.

# Executive One-Pager #2: Building an Open Source Enabling Environment

**What Executives Need to Be Aware of:**

* The key enabling factors and barriers to the adoption of open source software within your organization, especially those related to staffing and organizational structure
* The wide variety of job functions, professional profiles and skill sets involved in developing, implementing and maintaining open source software projects in a government organization
* Best practices for building and leading a right-sized technical team

## Key Points

* **A growing policy and legislative landscape supports the adoption of open source software.** However, current structures, practices, and mentalities within many government and public sector agencies can create barriers when it comes to open source software adoption.
* **Technical staffing is one of the greatest perceived barriers to open source software adoption.** A lack of in-house capacity and collaborative culture can lead to over-reliance on vendors: Few government organizations have full-time staff (at leadership and implementation levels) with necessary skills and knowledge to procure, build, and/or maintain open source software. As a result, governments typically rely on vendors.[[89]](#footnote-89) This can cause vendor lock-in, ballooning costs and hidden fees, and loss of control (over functionalities, updates and compliance).[[90]](#footnote-90)
* **Executives in organizations of all sizes can overcome these challenges by making strategic investments, reducing real and perceived risk, and encouraging a cultural shift in the organization**. This includes:
  + Learning how open source software procurement, development, and maintenance works and supporting internal working groups and champions.
  + Driving an organizational culture that encourages working in the open and continuous user engagement.

Engaging with a variety of partners, peer organizations and service providers to complement the internal team.

# Executive One-Pager #3: Procurement, Contracting and Budgeting for Open Source Software

**What Executives Need to Be Aware of:**

* The key differences between licensing and development approaches and why government systems are biased toward proprietary software and waterfall development
* The different software revenue models and how they affect government cost structures
* New approaches to procurement, contracting, licensing and staffing and how these create value and advance digital government goals

## Key Points

* **Procurement, contracting and budgeting structures and norms are designed for pre-digital assets – and therefore make it challenging to obtain open source software.** However, as government organizations embrace digital-native approaches, open source software and agile method becomes the easiest path.
* **Open source software requires a different approach to budgeting than proprietary software.** While proprietary software is priced using one of many standard revenue models, open source software is freely distributed, but requires investments in staff capacity and/or service providers to deploy and maintain software.
* **Open source software aligns well with ‘agile’ methods** – which also happen to be the best practice for software development. When procuring custom software services, government should use modular contracting and budgeting approaches, which enable Agile development.
* **Procurement is a creative and exploratory process.** Using procurement correctly, you can not only find the best, lowest cost solution, but also generate positive spin-off effects.
* **Pooled purchasing is an effective strategy** for a peer group of government organizations to share the cost of obtaining and maintaining open source software that fills a common need.

# Executive One-Pager #4: Open Source Software Security and Maintenance

**What Executives Need to Be Aware of:**

* The interlinked security and maintenance considerations associated with both conventional proprietary and open source software.
* The key challenges that organizations face in maintaining and securing their open source software solutions
* Strategies and best practices for ensuring successful stewardship and maintenance of open source software

## Key Points

* **There is no such thing as zero-risk software: all software – proprietary and open source – is built on sub-modules and involves risks that must be managed.** In fact, most conventional proprietary companies use open source code in their products and participate in maintaining open source software.
* **Open source software is not inherently less secure than conventional proprietary software**. As with any software, open source software carries certain risks that are unavoidable, but this should not discourage you from considering it for your organization. The best course of action is to conduct a thorough risk assessment and make appropriate plans to mitigate those risks.
* **More ‘eyes’ on a piece of open source software increase the chances that critical maintenance issues and security vulnerabilities will be resolved.** Open source software that is backed by a large active community of users and developers is more likely to receive regular updates and security patches. However, if you are using open source software that is not widely supported by a community of users, you will need to compensate by making sure you have appropriate in-house capacity (or can hire external support) to monitor for security vulnerabilities.
* **Don’t fork it unless you can maintain it; the more bespoke code your organization creates, the more the maintenance burden falls on you**. If you create an independent local instance (forking the code) updates and security patches are your own responsibility – and they become increasingly difficult, as your version and the main branch grow further apart over time.

# Executive One-Pager #5: Collaborating on Open Source Software

**What Executives Need to Be Aware of:**

* Key considerations when forming multi-jurisdictional teams around a particular piece of software, including resource-sharing between different jurisdictions or levels of government
* What it means to participate in a global community of codebase stewardship
* What it means to release software under an open source license – and the value in doing so

## Key Points

* **Collaboration on open source software is a win-win.** The software you build, or the adaptations you make, will be useful to other jurisdictions as well. Open source software performs better and becomes more secure as the community around it grows. It is therefore in each user’s best interest to contribute, and to add more contributing users.
* **Government organizations needn’t reinvent the wheel.** When source code is released under an open source license, any jurisdiction can make small software adaptations to ensure that the software is best-fit to local use cases, regulations and standards. When a core set of functionalities are shared, each jurisdiction’s time, effort and resources can be focused on making small adaptations to fit the software to their local context (rather than building a full set of core functionalities from scratch).
* **Same-level governments are natural collaborators when it comes to designing, obtaining and maintaining open source software.** They are responsible for delivering a similar set of services and have a similar set of needs (internal, administrative and external, resident-facing). They are therefore natural collaborators when it comes to designing, obtaining and maintaining open source software.
* **Jurisdictions already collaborate in various well-established networks and peer groups that can be activated as open source software stewardship communities.** Distributed version control platforms (like GitHub) enable users to contribute or suggest changes in a structured and collaborative way.

# 

# Appendix B: Implementing the Course

The learning materials can be distributed in their current form as a static document. If distributing as a webpage or e-learning course, please note the following considerations:

* We recommend using the [Course Overview](#_heading=h.k7bllgkqq56i) section as a landing page. The [Front Matter](#_heading=h.fcmx27u5o5w9), [Glossary](#_heading=h.zu0gcz), and [Executive One-Pagers](#_heading=h.hzug8chicr9f) (1-2 pagers) can also be linked on this page.
* Citations have been provided as footnotes in order to maintain their relationship to the material cited if content is reorganized in the future.
* Modules can be displayed as a single page (HTML) or divided into slides/sub-sections at the Heading 1 or Heading 2 level.
* Appendix B (i.e., this page) should be excluded from public-facing instances of this course.

1. Treasury Board of Canada Secretariat, “Open First Whitepaper: Open Markets,” July 28, 2020,<https://www.canada.ca/en/government/system/digital-government/digital-government-innovations/open-source-software/open-first-whitepaper/open-first-whitepaper-markets.html>. [↑](#footnote-ref-1)
2. Public Digital, “Open Source in Government: Creating the Conditions for Success,” June 2021,<https://public.digital/research>. [↑](#footnote-ref-2)
3. Matthew Claudel and Bianca Wylie, “Technology Procurement: Shaping Future Public Value,” Community Solutions Network Research Brief (Open North, 2021),<https://opennorth.ca/publications/technology-procurement-shaping-future-public-value/>. [↑](#footnote-ref-3)
4. “Open Source Software Requirements and Guidance (Draft),” HTML (2018; repr., Government of Canada, June 14, 2022),<https://github.com/canada-ca/open-source-logiciel-libre>. [↑](#footnote-ref-4)
5. Government of Ontario, “Digital Service Standard, 2021,” ontario.ca, accessed November 22, 2022, <http://www.ontario.ca/page/digital-service-standard>. [↑](#footnote-ref-5)
6. Gouvernement du Québec. Secrétariat du Conseil du trésor, “Logiciels Libres,” 2009,<https://www.tresor.gouv.qc.ca/ressources-informationnelles/logiciels-libres/>. [↑](#footnote-ref-6)
7. Gouvernement du Québec. Forge Gouvernementale, “Québec Free and Open Source Licence (LiLiQ),” accessed November 10, 2022,<https://forge.gouv.qc.ca/licence/en/>. [↑](#footnote-ref-7)
8. Apolitical, “How Opening up Government Source Code Improves Everything!,” Apolitical, December 7, 2021,<https://apolitical.co/solution-articles/en/how-opening-up-government-source-code-improves-everything>. [↑](#footnote-ref-8)
9. Office of the Privacy Commissioner of Canada, “Summary of Privacy Laws in Canada,” May 15, 2014,<https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/02_05_d_15/>. [↑](#footnote-ref-9)
10. Teaching Public Service in the Digital Age: The Digital Era Competencies

    <https://www.teachingpublicservice.digital/en/competencies> [↑](#footnote-ref-10)
11. Paulina Grzegorzewska, “Lessons from the Private Sector for Governmental OSPOs,” Joinup, February 9, 2021, <https://joinup.ec.europa.eu/collection/open-source-observatory-osor/news/lessons-private-sector-governmental-ospos>. [↑](#footnote-ref-11)
12. Paulina Grzegorzewska, “Lessons from the Private Sector for Governmental OSPOs,” Joinup, February 9, 2021, <https://joinup.ec.europa.eu/collection/open-source-observatory-osor/news/lessons-private-sector-governmental-ospos>. [↑](#footnote-ref-12)
13. New America, “Section Four: Managing the Details,” Building and Reusing Open Source Tools for Government, accessed July 4, 2022,<https://www.newamerica.org/digital-impact-governance-initiative/reports/building-and-reusing-open-source-tools-government/section-four-managing-the-details>. [↑](#footnote-ref-13)
14. [Public Digital, “Open Source in Government.”](https://www.zotero.org/google-docs/?ZHVaPD) [↑](#footnote-ref-14)
15. [Public Digital, “Open Source in Government.”](https://www.zotero.org/google-docs/?ZHVaPD) [↑](#footnote-ref-15)
16. Treasury Board of Canada Secretariat, “Open First Whitepaper,” July 28, 2020,<https://www.canada.ca/en/government/system/digital-government/digital-government-innovations/open-source-software/open-first-whitepaper.html>. [↑](#footnote-ref-16)
17. Aaron Burk and Rebecca Refoy, “A Day in the Life of an 18F Product Owner,” *18F* (blog), September 18, 2017,<https://18f.gsa.gov/2017/09/18/a-day-in-the-life-of-an-18f-product-owner/>. [↑](#footnote-ref-17)
18. “18F Partnership Principles,” 18F, accessed October 18, 2022,<https://18f.gsa.gov/partnership-principles/>. [↑](#footnote-ref-18)
19. New America, “Section Two: Building Open Source Software,” Building and Reusing Open Source Tools for Government, accessed July 4, 2022,<http://newamerica.org/digital-impact-governance-initiative/reports/building-and-reusing-open-source-tools-government/>. [↑](#footnote-ref-19)
20. Treasury Board of Canada Secretariat, “6. Build in Accessibility from the Start - Government of Canada Digital Playbook (Draft),” accessed October 18, 2022,<https://canada-ca.github.io/digital-playbook-guide-numerique/views-vues/standards-normes/en/6-build-in-accessibility-from-start.html>. [↑](#footnote-ref-20)
21. Shira Ovide, “Government Tech Moves Too Slooowly,” *The New York Times*, July 7, 2021,<https://www.nytimes.com/2021/07/07/technology/jedi-government-tech.html>. [↑](#footnote-ref-21)
22. Heather J. Meeker, “Chapter 13. Enforcement of Open Source Licenses,” in *The Open Source Alternative: Understanding Risks and Leveraging Opportunities* (Hoboken, NJ: Wiley & Sons, 2008),<https://www.oreilly.com/library/view/the-open-source/9780470194959/ch13.html#ftn.CHP-13-FN-145>. [↑](#footnote-ref-22)
23. AltexSoft, “Revenue Model Types and Examples,” June 12, 2020,<https://www.altexsoft.com/blog/revenue-model-types/>. [↑](#footnote-ref-23)
24. Waldo Jacquith, “The Disconnect between Software Development and Government Contracting,” May 8, 2021,<https://waldo.jaquith.org/blog/2021/05/disconnect-development-and-government-contracting/>. [↑](#footnote-ref-24)
25. Waldo Jacquith, “The Disconnect between Software Development and Government Contracting.” [↑](#footnote-ref-25)
26. Waldo Jacquith, “The Disconnect between Software Development and Government Contracting.” [↑](#footnote-ref-26)
27. Matthew Claudel and Bianca Wylie, “Technology Procurement: Shaping Future Public Value,” Community Solutions Network Research Brief (Open North, 2021),<https://opennorth.ca/publications/technology-procurement-shaping-future-public-value/>. [↑](#footnote-ref-27)
28. Matthew Claudel and Bianca Wylie, “Technology Procurement: Shaping Future Public Value,” Community Solutions Network Research Brief (Open North, 2021),<https://opennorth.ca/publications/technology-procurement-shaping-future-public-value/>. [↑](#footnote-ref-28)
29. Josh Lowe, “Canadian Budget Leads with Digital Tax and IT Investments,” *Global Government Forum*, April 20, 2021,<https://www.globalgovernmentforum.com/canadian-budget-leads-with-digital-tax-and-it-investments/>. [↑](#footnote-ref-29)
30. Business Development Bank of Canada, “Technology Industry Outlook: How Changes in the Economy Affect Canada’s Tech Sector,” January 2021,<https://www.bdc.ca/globalassets/digizuite/28336-st-outlookmfg-e2010-2.pdf>. [↑](#footnote-ref-30)
31. Luke DeCoste, “Outdated Procurement Rules Hindering Digital Government,” Policy Options, February 12, 2019,<https://policyoptions.irpp.org/magazines/february-2019/outdated-procurement-rules-hindering-digital-government/>. [↑](#footnote-ref-31)
32. Kaye Sklar, “A Procurement Path to Equity: Strategies for Government and the Business Ecosystem” (Open Contracting Partnership and Aspen Institute Center for Urban Innovation, 2020),<https://www.open-contracting.org/wp-content/uploads/2020/11/OCP-AspenCUI-2020-Pathway-to-Equity.pdf>. [↑](#footnote-ref-32)
33. Jerrod Larson, Jen Hocko, and Richard Bye, “User-Centered Procurement: Evaluating the Usability of ‘Off-the-Shelf’ Software User Experience Magazine,” *UX Development*, March 2010,<https://uxpamagazine.org/user_centered_procurement/>. [↑](#footnote-ref-33)
34. LaBrie, Suzanne and Ferron, Pierre-Antoine, “What is Open Procurement?” *Open North Community Solutions Network*, <https://lms.opennorth.ca/catalog/info/id:130> [↑](#footnote-ref-34)
35. While it only covers a handful of major cities, [Open Source Cities](https://github.com/opensourcecities) gives you a sense of the variety of actors and organizations involved in local open source ecosystems. Once you start looking, you may be surprised to learn how much open source activity is already happening in your area. [↑](#footnote-ref-35)
36. Sascha Haselmayer, “Unit 3: Choosing Your Procurement Path,” *Citymart Procurement Institute* (blog), March 26, 2020,<https://medium.com/citymartinsights/unit-3-choosing-your-procurement-path-8e1b711d0268>. [↑](#footnote-ref-36)
37. Ben Winter, “4 Key Steps to Performing an Effective Spend Analysis,” *Fairmarkit* (blog), August 25, 2021,<https://www.fairmarkit.com/blog/4-key-steps-to-performing-an-effective-spend-analysis>; Doug Greer and Jen Scarlato, “Best Practices for TCO Costing,”<https://cdn2.hubspot.net/hubfs/2652075/Downloadable_Files/regoUniversity%202018/Functional/TCO%20Best%20Practices%20for%20TCO%20Costing_SVMGT01.pdf>. [↑](#footnote-ref-37)
38. Treasury Board of Canada Secretariat, “Guide for Using Open Source Software,” “Evaluate Support Options,” July 28, 2020,<https://www.canada.ca/en/government/system/digital-government/digital-government-innovations/open-source-software/guide-for-using-open-source-software.html>. [↑](#footnote-ref-38)
39. Laura Gerhardt and Mark Headd, “Why We Love Modular Contracting,” *18F* (blog), April 9, 2019,<https://18f.gsa.gov/2019/04/09/why-we-love-modular-contracting/>. [↑](#footnote-ref-39)
40. Mark Hopson, “Improving Government Outcomes through an Agile Contract Format,” *18F* (blog), November 30, 2017,<https://18f.gsa.gov/2017/11/30/improving-government-outcomes-through-an-agile-contract-format/>. [↑](#footnote-ref-40)
41. Association of Certified Fraud Examiners, “Contract and Procurement Fraud: Fraud in Procurement without Competition,”<https://www.fraudconference.com/uploadedFiles/Fraud_Conference/Content/Course-Materials/presentations/23rd/ppt/post-06-Fraud-in-Procurement-Without-Competition.pdf>. [↑](#footnote-ref-41)
42. US General Services Administration, “2nd Generation IT Blanket Purchase Agreements,” <https://www.gsa.gov/technology/technology-purchasing-programs/mas-information-technology/buy-from-mas-information-technology/2nd-generation-it-blanket-purchase-agreements> [↑](#footnote-ref-42)
43. Rémy Bernard, “The Problems with Project-Based Funding for IT Organizations,” *Information Technology Strategy Team* (blog), May 18, 2021,<https://sara-sabr.github.io/ITStrategy/2021/05/18/problems-of-project-based-funding.html>. [↑](#footnote-ref-43)
44. Rémy Bernard, “The Problems with Project-Based Funding for IT Organizations.” [↑](#footnote-ref-44)
45. Rémy Bernard, “Funding Software Related Activities,” *Information Technology Strategy Team* (blog), July 21, 2022,<https://sara-sabr.github.io/ITStrategy/2022/07/21/funding-sw-activities.html>. [↑](#footnote-ref-45)
46. Zachary Spicer, “Digital Dilemmas: Technology, Governance, and Canadian Municipalities,” IMFG Forum Papers (Institute on Municipal Finance & Governance, 2022), 7-8<https://imfg.munkschool.utoronto.ca/research/doc/?doc_id=597>. [↑](#footnote-ref-46)
47. Ben Balter, “Eight Reasons Why Government Contractors Should Embrace Open Source Software,” *Ben Balter* (blog), October 8, 2014,<https://ben.balter.com/2014/10/08/why-government-contractors-should-embrace-open-source/>. [↑](#footnote-ref-47)
48. Maija Jokiniemi, “The European Commission Acknowledged Forum Virium Helsinki’s Public Procurement Tool as a Leading Innovation,” Forum Virium Helsinki, August 19, 2022,<https://forumvirium.fi/en/the-european-commission-acknowledged-forum-virium-helsinkis-public-procurement-tool-as-a-leading-innovation/>. [↑](#footnote-ref-48)
49. Chris Romeo, “A Security Practitioner’s Guide to Software Obsolescence,” TechBeacon, accessed September 29, 2022,<https://techbeacon.com/security/security-practitioners-guide-software-obsolescence>. [↑](#footnote-ref-49)
50. Keith Turpin, “Secure Coding Practices - Quick Reference Guide” (OWASP Foundation, 2010), [4 https://owasp.org/www-pdf-archive/OWASP\_SCP\_Quick\_Reference\_Guide\_v2.pdf](https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v2.pdf). [↑](#footnote-ref-50)
51. Nadia Eghbal, “Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure” (Ford Foundation, 2016), 77 <https://www.fordfoundation.org/work/learning/research-reports/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure/>. [↑](#footnote-ref-51)
52. Liam Tung, “Open Source Security Fears Are Fading Away,” ZDNET, March 3, 2022,<https://www.zdnet.com/article/is-open-source-as-proprietary-software-these-tech-chiefs-think-it-is/>. [↑](#footnote-ref-52)
53. Rafael Laguna, “4 Myths About Open Source We Should Put to Rest | WIRED,” March 2013,<https://www.wired.com/insights/2013/03/4-myths-about-open-source-to-put-to-rest/>. [↑](#footnote-ref-53)
54. Canadian Centre for Cyber Security, “National Cyber Threat Assessment 2020” (Government of Canada, 2020),<https://cyber.gc.ca/sites/default/files/cyber/publications/ncta-2020-e-web.pdf>. [↑](#footnote-ref-54)
55. Office of the Privacy Commissioner of Canada, “A Data Privacy Day Conversation with Canada’s Privacy Commissioner,” February 5, 2020,<https://www.priv.gc.ca/en/opc-news/speeches/2020/sp-d_20200128/>. [↑](#footnote-ref-55)
56. Canadian Centre for Cyber Security, “Security Considerations When Using Open Source Software (ITSAP.10.059),” Canadian Centre for Cyber Security, July 9, 2020,<https://www.cyber.gc.ca/en/guidance/security-considerations-when-using-open-source-software-itsap10059>. [↑](#footnote-ref-56)
57. Canadian Centre for Cyber Security, “Security Considerations When Using Open Source Software (ITSAP.10.059).” [↑](#footnote-ref-57)
58. Canadian Centre for Cyber Security, “Security Considerations When Using Open Source Software (ITSAP.10.059).” [↑](#footnote-ref-58)
59. Canadian Centre for Cyber Security, “Security Considerations When Using Open Source Software (ITSAP.10.059).” [↑](#footnote-ref-59)
60. Eric Brewer et al., “Know, Prevent, Fix: A Framework for Shifting the Discussion around Vulnerabilities in Open Source,” *Google Open Source Blog* (blog), February 3, 2021,<https://opensource.googleblog.com/2021/02/know-prevent-fix-framework-for-shifting-discussion-around-vulnerabilities-in-open-source.html>. [↑](#footnote-ref-60)
61. Jim Boehm et al., “The Risk-Based Approach to Cybersecurity,” McKinsey & Company, October 8, 2019,<https://www.mckinsey.com/capabilities/risk-and-resilience/our-insights/the-risk-based-approach-to-cybersecurity>. [↑](#footnote-ref-61)
62. Zeljka Zorz, “The Percentage of Open Source Code in Proprietary Apps Is Rising,” *Help Net Security* (blog), May 22, 2018,<https://www.helpnetsecurity.com/2018/05/22/open-source-code-security-risk/>. [↑](#footnote-ref-62)
63. Oskar Barcz, “What Is Dependency Hell and How to Avoid It?,” November 26, 2021,<https://www.boldare.com/blog/software-dependency-hell-what-is-it-and-how-to-avoid-it>. [↑](#footnote-ref-63)
64. The purpose of the left-pad package is very simple: it “pads” the left-hand side of a string (of integers or characters) by adding zeroes or spaces to reach a desired string length. [↑](#footnote-ref-64)
65. Keith Collins, “How One Programmer Broke the Internet by Deleting a Tiny Piece of Code,” Quartz, March 27, 2016,<https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code/>. [↑](#footnote-ref-65)
66. David Haney, “NPM & Left-Pad: Have We Forgotten How To Program?,” David Haney, accessed August 29, 2022,<https://www.davidhaney.io/npm-left-pad-have-we-forgotten-how-to-program/>. [↑](#footnote-ref-66)
67. United States Cybersecurity & Infrastructure Security Agency, “Software Bill of Materials,” accessed November 13, 2022,<https://www.cisa.gov/sbom>. [↑](#footnote-ref-67)
68. New America, “Section Two: Building Open Source Software,” Building and Reusing Open Source Tools for Government, accessed July 4, 2022,<http://newamerica.org/digital-impact-governance-initiative/reports/building-and-reusing-open-source-tools-government/>. [↑](#footnote-ref-68)
69. Treasury Board of Canada Secretariat, “Guide for Using Open Source Software.” [↑](#footnote-ref-69)
70. Ann Cavoukian and Mark Dixon, “Privacy and Security by Design: An Enterprise Architecture Approach” (Information and Privacy Commissioner of Ontario, 2013),<https://www.ipc.on.ca/wp-content/uploads/Resources/pbd-privacy-and-security-by-design-oracle.pdf>. [↑](#footnote-ref-70)
71. Nicole Olsen, “Open Source Projects and the GDPR,” Privacy Policies, accessed September 2, 2022,<https://www.privacypolicies.com/blog/gdpr-open-source-projects/>. [↑](#footnote-ref-71)
72. Treasury Board of Canada Secretariat, “Guide for Using Open Source Software,” July 28, 2020,<https://www.canada.ca/en/government/system/digital-government/digital-government-innovations/open-source-software/guide-for-using-open-source-software.html>. [↑](#footnote-ref-72)
73. Open Source Security Foundation and The Linux Foundation, “Home,” OpenSSF Scorecard, accessed November 14, 2022,<https://securityscorecards.dev/undefined/>. [↑](#footnote-ref-73)
74. Treasury Board of Canada Secretariat, “Guide for Using Open Source Software.” [↑](#footnote-ref-74)
75. Justin C. Colannino, “Free and Open Source Software in Municipal Procurement: The Challenges and Benefits of Cooperation,” *Fordham Urban Law Journal* 39 (February 2016): 903–29,<https://ir.lawnet.fordham.edu/ulj/vol39/iss4/2>. [↑](#footnote-ref-75)
76. Treasury Board of Canada Secretariat, “Guide for Using Open Source Software,” July 28, 2020,<https://www.canada.ca/en/government/system/digital-government/digital-government-innovations/open-source-software/guide-for-using-open-source-software.html>. [↑](#footnote-ref-76)
77. Guide for Publishing Open Source Software. <https://www.canada.ca/en/government/system/digital-government/digital-government-innovations/open-source-software/guide-for-publishing-open-source-code.html> [↑](#footnote-ref-77)
78. Ben Balter, “Everything a Government Attorney Needs to Know about Open Source Software Licensing,” *Ben Balter* (blog), October 8, 2014,<https://ben.balter.com/2014/10/08/open-source-licensing-for-government-attorneys/>; GitHub, “Choose an Open Source License,” Choose a License, accessed September 6, 2022,<https://choosealicense.com/>; GitHub, “The Legal Side of Open Source,” Open Source Guides, October 17, 2022,<https://opensource.guide/legal/>. [↑](#footnote-ref-78)
79. Treasury Board of Canada Secretariat, “Guide for Using Open Source Software,” July 28, 2020,<https://www.canada.ca/en/government/system/digital-government/digital-government-innovations/open-source-software/guide-for-using-open-source-software.html>. [↑](#footnote-ref-79)
80. Margaret Lin, “Learning from Failure: When Sharing Software Doesn’t Work,” *Beeck Center* (blog), April 11, 2022,<https://beeckcenter.georgetown.edu/learning-from-failure-when-sharing-software-doesnt-work/>. [↑](#footnote-ref-80)
81. Waldo Jaquith and Robin Carnahan, “Sharing Government Software: How Agencies Are Cooperatively Building Mission-Critical Software” (Beeck Center for Social Impact + Innovation, Georgetown University, 2021),<https://beeckcenter.georgetown.edu/wp-content/uploads/2021/04/Sharing-Government-Software.pdf>. [↑](#footnote-ref-81)
82. Margaret Lin, “Learning from Failure: When Sharing Software Doesn’t Work,” *Beeck Center* (blog), April 11, 2022,<https://beeckcenter.georgetown.edu/learning-from-failure-when-sharing-software-doesnt-work/>. [↑](#footnote-ref-82)
83. Foundation for Public Code, “Governance Game,” accessed November 3, 2022,<https://about.publiccode.net/activities/supporting-codebase-governance/game/>. [↑](#footnote-ref-83)
84. Miller, Ben. “How Government Is Reforming IT Procurement and What it Means for Vendors,” *GovTech,* April 6, 2017. <https://www.govtech.com/biz/how-government-is-reforming-it-procurement-and-what-it-means-for-vendors.html> [↑](#footnote-ref-84)
85. Center for Global Development, “Better Together: Exploring the Role of Pooled Procurement in Improving Access to Medicines amid COVID-19,” <https://www.cgdev.org/blog/better-together-exploring-role-pooled-procurement-improving-access-medicines-amid-covid-19> [↑](#footnote-ref-85)
86. CoProcure, “What is Cooperative Procurement?” <https://www.coprocure.us/blog/cooperative-contracts-joint-solicitation-vs-piggybacking/> [↑](#footnote-ref-86)
87. National Association of State Procurement Officials (NASPO), “Strength in Numbers: An Introduction to Cooperative Procurements,” <https://naspovaluepoint.org/wp-content/uploads/2020/08/Cooperative_Purchasing0410update.pdf> [↑](#footnote-ref-87)
88. Foundation for Public Code, “Introduction, Standard for Public Code,” September 7, 2022,<https://standard.publiccode.net/introduction.html>. [↑](#footnote-ref-88)
89. Public Digital, “Open Source in Government: Creating the Conditions for Success,” June 2021,<https://public.digital/research>. [↑](#footnote-ref-89)
90. Matthew Claudel and Bianca Wylie, “Technology Procurement: Shaping Future Public Value,” Community Solutions Network Research Brief (Open North, 2021),<https://opennorth.ca/publications/technology-procurement-shaping-future-public-value/>. [↑](#footnote-ref-90)