

Learning Materials on the Use of Open Source Software in Government

Draft Content

Open North Inc.

September 19, 2022

Prepared for:

The Institute for Citizen-Centred Service

The Joint Councils



OpenNorth

Table of Contents

Table of Contents	2
Course Overview.....	5
Course Learning Objectives	5
Overview of Course Modules	5
Module 1: Introduction to Open Source Software.....	6
Learning Objectives	6
Introduction	6
Key Takeaways.....	6
Why Open Source Software Matters in Government.....	7
Section 1: How Do Governments Use Software?	7
Software in Government: Use Cases	7
How Government Software is Unique.....	7
Section 2: Software Basics	8
Software Basics: Key ideas and terms	8
A (Brief) Overview of Software Licensing Models	8
Section 3: Proprietary vs. Open Source Software.....	9
The Basics of Proprietary Software.....	9
The Origin of the Open Source Software Movement.....	9
Open Source Software Licensing.....	10
Open Source Software Development.....	10
Open Source Software Contractors	11
Budgeting for OSS and Working with Contractors	11
Section 4: Benefits and Challenges of OSS in Government.....	12
The Unique Benefits of Open Source Software in Government.....	12
The Challenges of Using Open Source Software in Government	12
Canadian Policy Landscape Related to Open Source Software	13
Example: CKAN in British Columbia	15
Summary	15
Further Resources.....	15
One-Page Module Summary	17

The state of Open Source Software in Canadian government	17
How does Open Source Software work?.....	17
Managing Open Source Software in government.....	17
Benefits to using Open Source Software in the public sector context.	17
Module 2: The Open Source Team	19
Learning Objectives	19
Introduction	19
Key Takeaways.....	19
Section 1: Components of an Open Source Team	19
Setting Up and Leading an Open Source Team	19
Product Managers.....	20
Further Resources.....	20
Module 3: Procurement, Contracting & Budgeting	21
Learning Objectives	21
Introduction	21
Key Takeaways.....	21
Section 1: Software Licensing, Revenue and Development Models	22
Software Licensing	22
Open Source versus Proprietary Software Licenses	22
A Brief History of Software Revenue Models.....	22
Proprietary Software Revenue Models	23
A (Brief) History of Software Development Models	23
Government Systems Prefer Waterfall Development.....	23
The Disconnect Between Software Procurement and Software Development.....	24
Matching Best Practice in Procurement and Software	24
Section 2: Digitally-Native Procurement.....	24
Seizing the Opportunity of Procurement	24
Overview of the Software Procurement Process.....	25
Managing the Procurement Process	25
Discovery: Market Research.....	26
Case Study: Mexico City's Bike Share System	26
Sourcing Strategies.....	27
True Lifetime Cost and Value of Ownership Analysis	27
Section 3: Contracting	28
Mitigating Downstream Contract Issues.....	28

Modular Contracting for Agile Software Development	28
Budgeting for Open Source Software	29
Further Resources	30
Module 4: Security & Privacy	31
Learning Objectives	31
Introduction	31
Key Takeaways.....	31
Section 1: Security and Open Source Software	31
Myths and Misconceptions about Open Source Software Security.....	31
Security Risks	32
Software Dependencies.....	32
Section 2: Privacy and OSS	33
What is Privacy?	33
Privacy in a Digital Age	34
The Legislative and Policy Context for Privacy.....	34
Addressing Privacy Risks Within OSS	35
Section 3: Assessing and Mitigating Security and Privacy Risks.....	36
Protecting Your Organization from OSS Security Risks	36
Further Resources.....	36
Module 5: Accessibility & Maintenance of Open Source Software	37
Learning Objectives	37
Introduction	37
Key Takeaways.....	37
Further Resources.....	37
Module 6: Working with Open Source Software Across Jurisdictions	38
Learning Objectives	38
Introduction	38
Key Takeaways.....	38
Further Resources.....	38
References	39
Glossary	40

Course Overview

The purpose of this course is to empower you and your organization by equipping you with the ideas, tools and resources you need to overcome the challenges that are often associated with using open source software (OSS) and thereby take advantage of its benefits.

Course Learning Objectives

By taking this course, you will:

1. Develop basic knowledge of Open Source Software fundamentals, how it differs from conventional proprietary software, and what value OSS brings in a public sector context.
2. Learn to recognize the common barriers to using OSS 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.

Overview of Course Modules

Course modules will address specific aspects of using and working with OSS in government:

- [Module 1: Introduction to Open Source Software](#)
- [Module 2: The Open Source Team](#)
- [Module 3: Procurement, Contracting & Budgeting](#)
- [Module 4: Security & Privacy](#)
- [Module 5: Accessibility & Maintenance](#)
- [Module 6: Working with OSS Across Jurisdictions](#)

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.
- Introduce Open Source Software and highlights that licensing is the fundamental difference between OSS and conventional proprietary software.
- Explore the differences between OSS and proprietary software. You will learn the origins and motivations of the Open Source Software movement, as well as the fundamentals of how OSS works today, and what it means to obtain and maintain OSS.
- Highlight the benefits of using OSS 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.

Introduction

Key Takeaways

- Many governments and public sector agencies across Canada and around the world are already using Open Source Software in major ways.
- 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.
 - How it is financed and how the revenue model works.
- OSS is freely distributed under an open-source license. It is continually maintained by a global community of users and developers. While the software itself is free, your organization may contract with a vendor to add specific features or do a custom integration with your existing digital environment.
- You don't have to build from scratch. With hundreds of full Open Source Software packages out there, and even more small component modules. The ecosystem offers as many options and functionalities as conventional proprietary software. You can use existing resources to establish a foundation, and then build on top of it.
- There are unique barriers to adopting OSS that exist in government – specific barriers, such as procurement processes and contract templates, and general barriers, such as norms and expectations.

Why Open Source Software Matters in Government

Open Source Software is well suited to the public sector context for key reasons. When using OSS, governments can:

1. Foster openness and transparency, in keeping with GC policy.
2. Receive constant updates that ensure performance and security.
3. Maintain direct control over costs.
4. Support the local economy and tech sector.
5. Avoid vendor lock-in and hidden costs.
6. Maintain direct control over software features and use, and maintain the right to adapt in response to changing policy requirements and standards.
7. Share resources and engage in peer-to-peer support networks.
8. Avoid “reinventing the wheel.” Build digital resources quickly by existing modules, so that you can focus on the specific requirements and constraints of the local context.
9. Reduce government liability through OSS licenses as opposed to proprietary licenses and enterprise contracts.

Section 1: How Do Governments Use Software?

Software in Government: Use Cases

Governments use software for a wide range of purposes, from citizen-facing services to procurement management to secure email. These use cases include:

- Translation and Accessibility (Audio captioning, Live feeds)
- Traffic and Parking Management (Dynamic pricing, Payment collection)
- Sewer and Wastewater Management (Real-time monitoring, Forecasting, Emergency alerts)
- Procurement Management (Issuing tenders, Vendor relations)
- Fees and Payments (Accepting payments for taxes, tickets, licenses, etc.)
- Data Management (Open data hosting)
- HR Management (Payroll, Hiring, Benefits)

How Government Software is Unique

Government is a unique context. The process of obtaining software – and the design of software itself – for government has distinctive features:

- It is sourced according to government procurement laws.
- It cannot fail or have gaps in service delivery.
- It often deals with private or secure information.
- Like government purchases in general, it is budgeted for up front, typically years in

advance. In today's best practice, the software development process is difficult to plan far in advance.

- Government organizations often put a representative in charge of a software project (a Contracting Officer Representative). This person may or may not be an end user or have a technical background.
- Many governments use legacy IT systems and digital environments. Maintaining these, and integrating contemporary software, is challenging and expensive.
- Government software must comply with additional regulations and standards:
 - Hosting software or data on local servers
 - Language and accessibility requirements
 - Choice justification (lengthy descriptions and approvals for specific coding languages, reusing existing code modules, features, etc.)¹

Section 2: Software Basics

Software Basics: Key ideas and terms

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* (writing lines of code). Software can be built from scratch, and/or composed of pre-existing modules (which perform specific functions). 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. For the purposes of this course, development also encompasses the ongoing updating and maintenance of software.
- **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 OSS.
- **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.

A (Brief) Overview of Software Licensing Models

- **Proprietary** refers to software for which the source code is owned by the individual or company who 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

¹ Anna Papadopoulos, “The Challenges of Developing Software for the Government,” *CEOWORLD Magazine*, July 7, 2020, <https://ceoworld.biz/2020/07/07/the-challenges-of-developing-software-for-the-government/>.

(enterprise resource planning software), ArcGIS (geographic information system).

- **Open source** refers to software that is available for anyone to access, use or change the source code.² 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).

Section 3: Proprietary vs. Open Source Software

The Basics of Proprietary Software

Conventional proprietary software is fundamentally different from OSS.

- **Development:** Software vendors build and commercialize software. Some software vendors will provide custom integrations with your organization's existing digital environment; if not, you will need to contract a service provider. 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). Software companies typically offer support but may charge fees.
- **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).
- **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 end users pay for.

The Origin of the Open Source Software Movement

Open Source Software as a community and a movement inspired by the opportunity of bringing 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 is zero. In this paradigm, the software was given away at no cost, but the rights stayed with the original developer.

² OpenLogic by Perforce, "Proprietary vs. Open Source," *OpenLogic by Perforce* (blog), August 29, 2019, <https://www.openlogic.com/blog/proprietary-vs-open-source>; *Open Source Basics* | Intel Software, 2014, <https://www.youtube.com/watch?v=Tyd0FO0tko8>; *What Is Open Source Explained in LEGO*, 2014, <https://www.youtube.com/watch?v=a8fHgx9mE5U>; *The Mind behind Linux* | Linus Torvalds, 2016, <https://www.youtube.com/watch?v=o8NPllzkFhE>.

- 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. 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.”³ Free software is like free beer (no cost), while Open Source Software is like free speech (unrestricted access and reproduction).

Open Source Software Licensing

Licensing is what makes OSS open.

- Anyone can develop software code and apply whatever license protection they like.
- 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 most common examples of Open Source licenses are [Creative Commons](#), the [GNU General Public License](#), and the [MIT License](#). These licenses allow the creator of a work to specify the terms of its use, reproduction and commercialization.
- Organizations like the [Free Software Foundation](#) and the [Software Freedom Law Center](#) 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.”⁴
- Some governments have published protocols for using Github, or frameworks for choosing an appropriate license.⁵

Open Source Software Development

Open Source Software is developed in an open, collaborative way.

- Like all software, OSS consists of lines of code. Usually, a working software package is a compilation of existing code modules that do specific things – like “sign-in authentication” – and those modules may also be Open Source Software.
- OSS is created and maintained in a public online repository that enables version control.
- These repositories use a contribution management protocol called “git” and the most prominent repository is called Github. Github allows a global community to manage,

³ Lessig, Lawrence. “Free, as in Beer.” *WIRED*, September 1, 2006. <https://www.wired.com/2006/09/free-as-in-beer/>.

⁴ 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>.

⁵ [BC Policy Framework for Github](#)

harmonize and reconcile simultaneous (possibly conflicting) versions of the software, identify and fix bugs, and streamline the process of editing.

- Because the license is open, any individual or organization can suggest contributions to the software's "source code." That means the software is continually improving through contributions from an open community. These are like branches on a tree.
- For most OSS, a governance structure is 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.
- Alternatively, 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. This is known as a "hard fork" because it starts a new path, disconnected from the original source code.

Open Source Software Contractors

Not every user of Open Source Software needs to be a proficient coder who participates in the global community.

- A software user, like a business, municipality or provincial government, can contract a third-party software service provider to work with OSS. In this case, the software itself is free, but there are costs associated with the development, integration, and maintenance work.
- This can take several forms:
 - **If there is no existing software that fills the need**, the contractor creates software from scratch and releases it under an Open Source license.
 - **If there is an existing OSS that needs to be altered**, the contractor creates customized features for the user.
 - **If there is an existing OSS that fills the needs**, the contractor deploys a custom instance that is integrated with the user's existing technical environment.

Budgeting for OSS and Working with Contractors

Budgeting for OSS should account for the shift from buying a product (proprietary software) to buying a service (development and integration work).

- This implies a shift in either or both *staffing* (having proficient coders on staff) and *procurement* (writing open contracts for services, as opposed to fixed costs for digital products and licenses).
- Get to know the local community of software developers. Having a good understanding of what these companies can do and what software they work with 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.
- It is important to develop strong, collaborative working relationships with software development contractors. Long-term contracts and repeated work allow contractors to become familiar with your technical environment and specific needs.

Section 4: Benefits and Challenges of OSS in Government

The Unique Benefits of Open Source Software in Government.

There are five 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. The global and local community audit the code and hold each other accountable. As a result of openness, OSS is also more interoperable across a variety of digital systems.
2. **Continuous updates and security.** Because the code is open, security auditors and contributors can inspect the code for security risks or flaws. Full software packages and sub-modules are constantly updated. This ensures performance and cybersecurity in changing conditions. Commonly used scripts help the community audit and debug software.
3. **Controlled costs.** OSS 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.
4. **Strong local (tech) economy.** Small local software contractors are well-suited to work with OSS and build custom features or integrations for government organizations. In contrast, proprietary software is generally created and commercialized by monolithic software vendors that are often located abroad. By hiring small local firms, government can strengthen the local economy, particularly its tech sector.
5. **Freedom from vendor lock-in.** Proprietary software can be operationally or contractually entrenched in an organization, putting the government at a disadvantage in terms of freely and fairly choosing the best solution. OSS creates open markets for providers of all kinds of support.
6. **Adaptation 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. OSS, on the other hand, can be quickly adapted and shared across jurisdictions.
7. **Focus on what matters locally.** Every jurisdiction is unique, in terms of policy and existing digital environment. OSS allows you to focus on what matters locally. Several public sector organizations can share a core set of OSS modules and each pay a small fee for a local contractor to create custom features for their unique environment.
8. **Reduced liability.** There are no obligations on the mere use of Open Source Software. Furthermore, OSS 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 six 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.** OSS is a very different cost structure. The software 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 vendors create barriers to obtaining and using OSS or contracting OSS service providers.

2. **Security.** OSS is not inherently less secure, but the mechanisms for updating software, finding bugs and fixing them are different than in conventional proprietary software. “Whereas closed-source software security depends on an attempt to maintain an information imbalance between the “white hats” (computer security analysts) and the “black hats” (malicious computer hackers), OSS software security depends upon an open competitive process of finding and closing security holes.”⁶
3. **Lack of 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.
4. **Supplier availability.** Smaller municipalities and rural areas may not have a mature local ecosystem of technology companies that are capable of providing services. However, several agencies work across Canada, serving a broad swath of customers.
5. **Rate of change.** Because OSS is 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 OSS, it is important to evaluate the level of activity within the support community.
6. **Governance.** OSS requires active participation involving contributors and users from different organizations, which can create challenges in managing the responsibility for maintenance.

Canadian Policy Landscape Related to Open Source Software

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

- The Government of Canada’s “open by default” policy, as well as associated programmes, mandates, and legislation, all encourage the use of OSS.

⁶ [Open First Whitepaper: Open Source Software Use](#)

- The Government of Canada maintains an Open Resource Exchange,⁷ as well as a searchable catalog of Open Source Software that are used by different levels of Canadian public administrations.⁸
- The Government of Canada's *Mandatory Procedures for Enterprise Architecture Assessment* "provide that, where possible, open source software be used first." In a procurement process, government organizations should "actively and fairly consider Open Source Software," according to the Procedures.
- The Government of Canada "is also 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."⁹
- Additional policies at the federal level include:
 - Directive on Service and Digital¹⁰
 - Government of Canada Digital Standards: Playbook¹¹

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.

- British Columbia: Guidelines on the Use of Open Source Software¹²
- Québec: "advocates the use of free software when it is the best choice and will ensure that leaders have the necessary support for this purpose." The Open Source Software information resource includes a Reference Guide, Maturity Analysis Guide and Total Cost of Ownership Analysis Guide.¹³
- The Québec Free License is an Open Source Software license specifically designed for use in the public sector, with three degrees of restriction on use, application and modification. It is validated by the Open Source Initiative.¹⁴

Municipalities also have a significant degree of control over the specific software and services they use, and some have policies that promote the use of Open Source Software.

- City of Montreal Open Policy¹⁵

⁷ <https://code.open.canada.ca/en/index.html>

⁸ <https://code.open.canada.ca/en/open-source-softwares.html>

⁹ <https://github.com/canada-ca/open-source-logiciel-libre>

¹⁰ Treasury Board of Canada Secretariat, "Directive on Service and Digital," May 6, 2022, <https://www.tbs-sct.canada.ca/pol/doc-eng.aspx?id=32601>.

¹¹ Treasury Board of Canada Secretariat, "Government of Canada Digital Playbook," 2018, <https://www.canada.ca/en/government/system/digital-government/government-canada-digital-standards.html>

¹² [British Columbia: Guidelines on the Use of Open Source Software](#)

¹³ <https://www.tresor.gouv.qc.ca/ressources-informatiionnelles/logiciels-libres/>

¹⁴ <https://forge.gouv.qc.ca/licence/en/>

¹⁵ <https://github.com/VilledeMontreal/politique-libre>

- Montreal OSS Inventory¹⁶
- City of Vancouver Open Policy¹⁷
- City of Toronto Digital Infrastructure Plan¹⁸
- Toronto OSS Inventory¹⁹

Example: CKAN in British Columbia

The B.C. Data Catalogue is powered by CKAN open source software. The B.C. Government participates in this software project, sharing developments with a community of international users.

Developed by the non-profit Open Knowledge Foundation (OKFN) for the Government of the United Kingdom, CKAN is by governments at all levels around the world, including the Government of Canada, the B.C. Government and some B.C. local governments.

[More than 115 governments and institutions around the world use CKAN](#) as their data catalogues and because code enhancements are shared, everyone using the software benefit from creating a better product at a lower cost to individual users. Community knowledge is another advantage of this project: GitHub, mailing lists, IRC channels and forums connect a vibrant community of over 100 CKAN global developers. This developer community has helped customize and implement CKAN software to meet the B.C. Government's needs.

Summary

This module has introduced you to basic OSS concepts, including situating OSS within the broader context of how governments use software.

Subsequent modules will cover these topics in more depth as they relate to specific government functions and areas of interest.

Further Resources

- [Who's using GitHub?](#): 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.
- **Shared Services Canada (SSC)** "delivers digital services to Government of Canada organizations. We provide modern, secure and reliable IT services so federal organizations can deliver digital programs and services that meet Canadians needs."²⁰
- **The Canadian Digital Service (CDS)** "helps Federal public servants deliver better government services." CDS offers coaching and advice, a product suite, and guides.
- **Linux Professional Institute (LPI)** "is the global certification standard and career support organization for open source professionals. With more than 200,000 certification holders, it's the world's first and largest vendor-neutral Linux and open source

¹⁶ [Montreal OSS Inventory](#)

¹⁷ [CBC, "City of Vancouver Embraces Open Data, Standards and Source"](#)

¹⁸ [City of Toronto Digital Infrastructure Plan \(presentation\)](#)

¹⁹ [Toronto OSS Inventory](#)

²⁰ <https://www.canada.ca/en/shared-services.html>

certification body.”²¹ It is a nonprofit organization that provides training and certification, and supports OSS communities.

- **The Government of Canada Digital Standards: Playbook** “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.”²² This is a great place to get started on your journey with OSS.

²¹ <https://www.lpi.org/>

²² <https://www.canada.ca/en/government/system/digital-government/government-canada-digital-standards.html>

One-Page Module Summary

The state of Open Source Software in Canadian government

- The Government of Canada has created policies, protocols and resources that explicitly mandate and/or support the use of Open Source Software.
- OSS is already in use across government agencies today, including core operations and enterprise-grade functions.
- There is a large and growing number of OSS software available today, and it offers sophisticated functionality that is comparable to proprietary software.
- Federal agencies and subnational government entities should incorporate OSS into their operating procedures and policies as they carry out digital transformation.
- Working with OSS in government requires changes to procurement, budgeting, staffing, and operations / service delivery.

How does Open Source Software work?

- Under an Open Source license, software is free for anyone to use, edit and distribute.
- Edits are managed in a distributed version control system, so that any changes are vetted and approved by the community.
- Therefore, the development and maintenance of OSS depends on an active community of contributors.
- The cost of proprietary software is in subscriptions, seats, and service fees paid to a company that owns the software license. OSS itself is free, but there are costs associated with either hiring a service provider to customize, implement and maintain software, or paying in-house staff who implement and maintain the software.

Managing Open Source Software in government.

- When considering OSS, governments choose whether to rely on in-house capacity or outside support.
- Budgeting for OSS requires a shift from capital expense to operating expense.
- Governments can build custom software using existing OSS modules.
- If governments develop custom software, GC policies recommend that the code be released under an Open Source License ("public money = public code").

Benefits to using Open Source Software in the public sector context.

1. Foster openness and transparency, in keeping with GC policy.
2. Receive constant updates that ensure performance and security.
3. Maintain direct control over costs.
4. Support the local economy and tech sector.
5. Avoid vendor lock-in and hidden costs.
6. Maintain direct control over software features and use, and maintain the right to adapt in response to changing policy requirements and standards.

7. Share resources and engage in peer-to-peer support networks.
8. Avoid “reinventing the wheel.” Build digital resources quickly by existing modules, so that you can focus on the specific requirements and constraints of the local context.
9. Reduce government liability through OSS licenses as opposed to proprietary licenses and enterprise contracts.

DRAFT

Module 2: The Open Source Team

MODULE UNDER DEVELOPMENT

Learning Objectives

In this module we will:

- Discuss the wide variety of job functions, professional profiles and skillsets involved in implementing and maintaining OSS solutions in a government organization
- Review guidelines for hiring and leading a right-sized technical team
- Introduce strategies for sharing capabilities across different government jurisdictions

Introduction

Key Takeaways

- Organizations that have previously procured software through large enterprise products and contracts may lack staff with necessary skills and knowledge to work with OSS.
 - It may seem like you can get around building in-house capacity by licensing from vendors. However, even buying licenses can be dangerous without adequate knowledge of implications (see [Module 3: Procurement, Contracting & Budgeting](#)).
 - Without in-house skills and capabilities at leadership and technical levels, governments are more reliant on vendors.²³
- Government staff do not necessarily need to be software developers. However, at a minimum, staff should have the skillsets and supporting tools necessary to evaluate open source solutions.

Section 1: Components of an Open Source Team

Setting Up and Leading an Open Source Team

- Strong leadership is essential in any organization – especially government – in setting policies and promoting best practices that will ensure the success of open source adoption.
- However, this leadership should be formalized wherever possible; reliance on one or two individuals to champion OSS off the side of their desk is a risk as all their efforts may be undone if they leave the organization.
- In smaller governments or agencies, the person accountable for the organization's technology strategy may be best positioned to oversee OSS adoption.²⁴

²³ Public Digital, "Open Source in Government: Creating the Conditions for Success," June 2021, <https://public.digital/research>

²⁴ Public Digital, "Open Source in Government."

- In larger governments or agencies with substantial internal capacity, setting up a centralized open source office or team is a good step toward institutionalizing open source.²⁵

Product Managers

The central figure in an effective software procurement is the product owner (also called a “project manager” or “business analyst”). This person shepherds the procurement process. They work across existing business units and are independent from those units. They embed with a department (end users) and bring back specific requirements. They have analytical and design thinking skills.

“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://sara-sabr.github.io/ITStrategy/2021/10/15/product-management-part-2.html>

<https://18f.gsa.gov/2017/09/18/a-day-in-the-life-of-an-18f-product-owner/>

Further Resources

²⁵ Public Digital, “Open Source in Government.”

Module 3: Procurement, Contracting & Budgeting

Learning Objectives

- Review Open Source and Proprietary software licensing models.
- Understand the differences between Agile and Waterfall development approaches.
- Understand why government systems bias toward proprietary software and waterfall development.
- Understand the different software revenue models and how they affect government cost structures.
- Learn new approaches to procurement, contracting, licensing and staffing.

Introduction

In order to fulfill GC digital policy goals and meet government service delivery standards,²⁶ subnational governments will need to incorporate software into a wide variety of digital services. OSS is a valuable option to ensure quality, ongoing control, and resource-efficiency.

Government norms, processes, and conventions are based on pre-digital assets and vendor service models. When these are implemented today, they favor proprietary licenses (for existing software) and waterfall development (for custom software). They assume that technical expertise is external, and they do not account for peer sharing and communities of practice.

Public sector executives have an opportunity to embrace a digital-native approaches 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 OSS can help them achieve digital transformation goals within limited budgets.

A digital-native procurement process – and the choice to use OSS – both empower public sector employees to be in control of how their digital toolsets are designed, managed, and implemented.

Key Takeaways

²⁶ Rémy Bernard, “Implementing Product Management in the Federal Government,” *Information Technology Strategy Team* (blog), October 15, 2021, <https://sara-sabr.github.io/ITStrategy/2021/10/15/product-management-part-2.html>.

Section 1: Software Licensing, Revenue and Development Models

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: Open Source and Proprietary.

Open Source versus Proprietary Software Licenses

- 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.
- 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.

A Brief History of Software Revenue Models

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 app monetization in which a user may access the main product for free, but will be charged for additional functions, services, bonuses, plugins, or extensions.

- Skype, Evernote, Gitbook.

Hybrid pricing. Sometimes pricing plans are a mixture of more than one. So that 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.²⁷

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

Proprietary Software Revenue Models

There are several different fee structures for conventional proprietary software.

- **Flat Rate:** a fixed price billed once or monthly (more common for on-prem software)
- **Usage-based:** pay according to the volume of use (could be based on the amount of data, the number of modules, etc).
- **Per-user:** pay for each unique credential (e.g. \$100 per user per year)
- **Tiered:** different levels (such as gold, silver, bronze) are priced individually, which may correlate to the number of users (e.g. \$100 per user up to 100 users, \$80 per user for 100-200 users).
- **Per active user:** pay for each unique credential that is actively used in a given month.
- **Freemium:** receive a basic feature set for free, and pay for additional functionality.

A (Brief) History of Software Development Models

Waterfall development is when you lay out The Plan in full before you start: every detail, every feature, every meeting, every benchmark, and then you're wrong on most of them but forge ahead anyway, secure in the knowledge that following The Plan gives air cover in the likely event that the thing you're building doesn't work the way you expected it to when you made The Plan.

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 2-4 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.

Government Systems Prefer Waterfall Development

"It's the job of contracting officers to get government the best value for their money. That means being sure that they'll get precisely what they need, within budget and on time. Normally, the best way to do this for a *firm, fixed price*: buying X gizmos that meet Y spec for \$Z. Firm fixed price (FFP) is by far the preferred method of procuring stuff. And for good cause!

²⁷ AltexSoft, "Revenue Model Types and Examples," June 12, 2020, <https://www.altexsoft.com/blog/revenue-model-types/>.

Contracting officers are taught that nearly any other contract type (i.e., labor-hour, cost reimbursement, or time and materials) is more risky. And that's generally true."²⁸

The Disconnect Between Software Procurement and Software Development

Government purchasing norms and rules are based on fixed assets and services (street lights or snow removal). In this paradigm, it is straightforward to define specs, compare options based on responses to an RFP, and procure a product or service for a firm, fixed price. When it comes to software, this means purchasing off-the-shelf software or a waterfall custom development.

"That's what has led us, for decades, down this path of wildly over-specified software RFPs. FFP for complex software means you need to have hundreds of pages of specs, because you've got to specify exactly what you're getting for your money. That's seen as safe.

In the meantime, software developers gave up that kind of software development and moved to Agile. We know that if you define all that stuff up front, you're sunk. You'll build something bad that the users hate, but that sure does meet the spec.

This has left a fundamental disconnect between contracting practices (specifying everything is safest) and software development practices (specifying everything is dangerous)."²⁹

Matching Best Practice in Procurement and Software

"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."³⁰

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.

In subsequent modules, we will explore what this means for procuring development services to create a custom instance of Open Source Software. We will also provide best practices and tools for procurement and modular budgeting that enables effective agile software development.

Section 2: 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

²⁸ Waldo Jacquith, "The Disconnect between Software Development and Government Contracting," May 8, 2021, <https://waldo.jacquith.org/blog/2021/05/disconnect-development-and-government-contracting/>.

²⁹ Waldo Jacquith, "The Disconnect between Software Development and Government Contracting."

³⁰ Waldo Jacquith, "The Disconnect between Software Development and Government Contracting."

procurement offers, they can creatively meet broad public sector objectives, promote innovation (inside and outside of government) and drive economic development.

[Statistics about the size of government spending, IT, local tech ecosystem, etc.]

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. Using specific tools like an RFP, the procurement process can specify:

Vendor Criteria: Small, Local, Minority-owned, or Nonprofit vendors

Solution Criteria: Public ownership / 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: supporting the local ecosystem of technology service providers, sharing software with peer jurisdictions, and maintaining public control over how data is used and commercialized.

Overview of the Software Procurement Process

1. Discovery
 - a. Actions: Capacity-building, Discovery Research, Market Research, Strategic analysis
 - b. Output: Choice of a strategy
2. Planning: Documentation and resourcing
 - a. Actions: RFP-writing, Project resourcing
 - b. Output: RFP is issued, or Agile process is planned
3. Assessment: Evaluation, compliance and contracting
 - a. Actions: Vendor interactions, Bid Evaluation, Contracting
 - b. Output: Purchase order or Service contract
4. Implementation: Building and deploying
 - a. Actions: Agile development, deployment and integration, Training
 - b. Output: All features delivered, software is integrated, staff are trained

Managing the Procurement Process

Traditionally, procurement is managed by a procurement official, based on detailed specifications given by someone closer to the end use – such as an IT department manager. However, this one-way flow of information 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 through these four phases: the Product Owner. The Product Owner needn't be a technical expert, but should have a working knowledge of software development processes and user-centred design methods (see [Module X: The Open Source Software Team for more detail](#)).

Discovery: Market Research

It is important that you become familiar 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:

- Market research and neutral market analysis of existing commercial solutions
- Look at how other cities have addressed the same / similar problems
- Issue an informal call for ideas (in the civic tech community)
- Issue a formal Request for Information (RFI) (through city channels)
- Host an event related to the problem and invite community groups and vendors
- 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. To maintain integrity and avoid concerns, ensure that all vendor engagements are public, well documented, synthesized as key insights, and that all information is easily accessible.

“Many of the very companies that have been selling closed-source software to the GC over the last decades have taken drastic changes to their development practices and adoption OSS due to the rapid pace of the industry and now have different business models that support OSS.”³¹

Market Research Resources

- [Methods of Market Engagement](#)
- [CityMart Market Research Worksheet](#)
- [Boston Sample RFI](#)

Case Study: Mexico City's Bike Share System

In 2019, Mexico City sought to expand its bikeshare program, the contract for which had been awarded to the same vendor for over ten years. However, the newly elected municipal government, which came in with a mandate for increased transparency and accountability, wanted to do procurement differently this time around.³²

... “the team carried out new market research and vendor outreach strategies for the city. These tactics included the city’s first-ever Request for Information process (RFI), which yielded suggestions to improve the service from nine companies with experience operating in 35 cities. The vendors also suggested a revised finance model combining sponsorship and government subsidies – an idea which the city pursued for Ecobici, and contributes to the long term financial viability of the system. The city was also committed to sharing information with the community and gathering their input...

³¹ 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>.

³² Gavin Hayman and Oscar Hernández, “How Mexico City Levelled Its Procurement Playing Field,” Apolitical, June 5, 2020, <https://apolitical.co/solution-articles/en/how-mexico-city-levelled-its-procurement-playing-field>.

To support trust and transparency throughout the planning process, the city created a dedicated project website. This website launched at the start of the team's work, and became the central hub for sharing information throughout the entire process [of gathering information, writing a tender, engaging with vendors and evaluating bids]. The team updated this site on a regular basis with all the information that suppliers or residents might be interested in, including the results from the subsequent RFI and user feedback processes, upcoming and past events, and key project milestones."³³

Sourcing Strategies

A fair and objective market research and cost analysis will enable you to confidently choose a sourcing strategy. If market research revealed one or more commercially available software solutions, you will choose between options (1) and (2). If you confirmed that there are no existing software solutions for the problem at hand, you will choose between options (3), (4) and (5).

You might:

1. Procure existing proprietary software
2. Adapt and implement existing Open Source Software in-house
3. Build custom software in-house
4. Contract a vendor to build custom software (then release it under an Open Source license)
5. Contract a vendor to adapt existing Open Source Software

More information on releasing custom software under an Open Source License **in Section X.**

[CityMart: Choosing your procurement path](#)

True Lifetime Cost and Value of Ownership Analysis

In order to weigh sourcing options, you must perform a True Lifetime Cost and Value of Ownership Analysis, including direct and indirect, immediate and long-term costs and benefits. This should include:

1. The direct cost of the solution (annual license or ownership)
2. The technical capacity of staff to build, adapt, integrate and/or maintain software
3. The acquisition costs, including all costs of the procurement process and change management (if applicable).
4. The operating costs, including annual fees, per-user cost, and insurance.
5. The cost of integration with existing digital infrastructure at the city (in terms of staff time and any software or database costs).
6. 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.

³³ Kaye Sklar, Oscar Hernández, and Mariana Lopez, "Peddling Openness: Mexico City's Journey to Procure a Better Bike Share System," *Open Contracting Partnership* (blog), January 27, 2022, <https://www.open-contracting.org/2022/01/27/peddling-openness-mexico-citys-journey-to-procure-a-better-bike-share-system/>.

7. The cost of future software customization (work orders) if the city's needs, processes, or underlying digital infrastructure changes in the future.
8. The value of controlling costs, revenue models, updates and additional feature sets (especially for compliance with changing government software performance standards).
9. The value of staff capacity-building and empowerment.
10. The value of releasing code under an Open Source license and engaging with a community of peer jurisdictions.

Total Cost of Ownership (TCO)

[Best Practices for TCO Costing | HubSpot](#)

FairMarkit: 4 Key steps to performing an effective spend analysis

<https://www.fairmarkit.com/blog/4-key-steps-to-performing-an-effective-spend-analysis>

Section 3: Contracting

Mitigating Downstream Contract Issues

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:** Proprietary software; Dependence on proprietary systems (like databases), Data use and ownership; Software ownership.
- **Security:** Vulnerability; Risk; Responsibility.
- **Interoperability:** APIs; Compatibility with existing technical infrastructure.
- **Revenue model:** Fee per "seat;" Fee per use; Freemium, etc. Be cautious of hidden fees or potential usage patterns that cause ballooning costs.
- **Maintenance:** Regular software updating; (Re)integration with new digital infrastructure; Ongoing user support.

Modular Contracting for Agile Software Development

Agile development is based on iterative short sprints that focus on building self-contained but interoperable modules. *Modular contracting* is a useful way of aligning budgets and deliverables with an agile development process. "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."³⁴ Each module should have an individual contract with goals, evaluation criteria, timeline and budget.

However, be cautious of *bid splitting*. This is 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](#)). Although auditing and compliance officials are

³⁴ 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/>.

becoming more and more accustomed to modular contracting for agile development, it is nevertheless important to preempt this concern.

Consider establishing an umbrella contract for the project through due process and with appropriate approvals. Then modularize the work and write sub-contracts for specific components. Ensure that the contract amount is a legitimate reflection of the work involved (rather than targeting a number just under the procurement threshold). 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 can integrate with the prior modules.

- 18F: Modular Contracting and Agile Development: <https://github.com/18F/Modular-Contracting-And-Agile-Development>
- 18F: Improving government outcomes through an agile contract format: <https://18f.gsa.gov/2017/11/30/improving-government-outcomes-through-an-agile-contract-format/>
- 18F: Why we love modular contracting: <https://18f.gsa.gov/2019/04/09/why-we-love-modular-contracting/>

Budgeting for Open Source Software

From Capital Expenditure (CapEx) to Operating Expenditure (OpEx)

Open Source Software is free, but it takes work to customize and maintain. That means the cost structure will be different: a shift from CapEx to OpEx. Government organizations will need to hire staff who have capacity to implement and maintain software together with the developer community, or contract technology service providers.

Modular Budgeting for Agile Development

Agile development is structured in short sprints, each focused on building a standalone module. Modular budgeting is an approach to financial planning that supports this working cadence. There is an umbrella contract, under which each module has specific, discrete deliverables that functionally stand on their own, as well as evaluation criteria and timeline. A modular budget applies to the person-hours and other costs associated with a discrete module.

If a vendor is under-performing, the subsequent modules can be transitioned to another quickly and easily. Modular budgets put you in control, so you can eliminate the risk.

Product-based Funding

Contemporary software – and OSS in particular – will draw on several different business units for development and ongoing maintenance. In many cases, it will be used by different business units, but at a minimum, software should be interoperable with IT systems across the city. That makes it hard

Budget for portfolios of projects

“Products are grouped into portfolios and managed by dedicated value stream teams. Teams are responsible for both the development and operation of the products. Budgeting is not done at the product level, it’s done at the team level (who will most likely manage more than one product in the portfolio).” This aligns well with the use of OSS – a well-budgeted and managed team can implement and maintain a large portfolio of software products.

Resourcing the IT team

In a government context, IT professionals do not necessarily work for a single agency or program – they often work across business units. Resourcing a product team must account for direct (labor, software subscriptions and licenses) and indirect (common resources) costing.

Pooled Purchasing

Procuring OSS on your own isn’t always feasible, especially for small and mid-sized municipalities. With pooled procurement you don’t need to bear the cost alone. There are several tools and strategies you can use to organize pooled purchasing among a group of organizations.

Further 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.

<https://www.open-contracting.org/>

Module 4: Security & Privacy

MODULE UNDER DEVELOPMENT

Learning Objectives

In this module we will:

- Discuss key security and privacy considerations associated with OSS
- Address commonly held perceptions (and misconceptions) regarding the risks of OSS compared to commercial proprietary software
- Explore various tools and techniques for assessing and mitigating security and privacy risks within OSS

Introduction

Key Takeaways

- **All software is built on other software.** Security is a key consideration for all software.
- **Open source software is not inherently less secure than conventional proprietary software.** However, the mechanisms for updating software, finding bugs and fixing them are different than in conventional proprietary software. “Whereas closed-source software security depends on an attempt to maintain an information imbalance between the “white hats” (computer security analysts) and the “black hats” (malicious computer hackers), OSS software security depends upon an open competitive process of finding and closing security holes.”³⁵
- **Continuous updates and security.** Because the code is open, security auditors and contributors can inspect the code for security risks or flaws. Full software packages and sub-modules are constantly updated. This ensures performance and cybersecurity in changing conditions. Commonly used scripts help the community audit and debug software.

Section 1: Security and Open Source Software

Myths and Misconceptions about Open Source Software Security

- **Open source software is a risk because it isn't secure.**
 - “In short, licensing models have nothing to do with security. If anything, OSS allows for development teams to rapidly address any security issues, whereas proprietary systems must rely on the software vendor to get around to issuing an update or patch. Additionally, open source projects benefit greatly from having a diverse community that is interested in the development of the solution, not just

³⁵ [Open First Whitepaper: Open Source Software Use](#)

one vendor. This allows for things like peer review from a base of knowledgeable and expert supporters. This really can't be undervalued, because while these communities are invested in the quality of the software, they aren't necessarily invested in making a buck from every functional improvement. In the end, if you asked a group of people to use the same processes to produce a proprietary model and OSS model of their software, they would likely be just as secure, and just as functional."³⁶

- **Open source software is a risk because you can't trust the people writing it.**

Security Risks

- **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.³⁷
- **Lack of verification:** The nature of how OSS is developed means that quality assurance and security testing are not guaranteed.³⁸
- **Lack of support:** No dedicated support team means that updates and security patches must be developed by the open source project community, which may take time.³⁹

Software Dependencies

All software is built on other software in one form or another. One benefit of this is that it allows developers to build new applications much faster than if they were writing code from scratch.

A **software dependency** is a code library or package that is reused in a new piece of software. For example, a machine learning project might call a Python library to build models.⁴⁰

When a code library or package critical to the functioning of a given piece of software is altered or deleted, the downstream impact is called a **dependency failure**.

Example:

- In 2016, a code package known as "[left-pad](#)"⁴¹ which consisted of just 11 lines of code was removed by its developer from the NPM Registry where it had been hosted. Its

³⁶ 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/>.

³⁷ 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>.

³⁸ Canadian Centre for Cyber Security, "Security Considerations When Using Open Source Software (ITSAP.10.059)."

³⁹ Canadian Centre for Cyber Security, "Security Considerations When Using Open Source Software (ITSAP.10.059)."

⁴⁰ Snyk, "6 Tips for Managing Your Open Source Components | Snyk," accessed August 29, 2022, <https://snyk.io/series/open-source-security/managing-open-source-components/>.

⁴¹ 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.

removal caused a cascade of dependency failures, resulting in software crashes worldwide.⁴²

- 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.”⁴³

The first step toward managing dependencies – and mitigating the impact of dependency failures – is simply to be aware of them.

Section 2: Privacy and OSS

What is Privacy?

In addition to being a pillar of our freedom, privacy has long been a constitutionally protected right under the Canadian Charter that is considered essential for exercising other rights.⁴⁴

- The individual right to privacy has been interpreted in Canadian case law as including protections for “personal privacy, territorial privacy, and informational privacy”.⁴⁵ Information privacy refers to the individual right to control with whom, how much, and for what purpose personal information is disclosed.⁴⁶
- Information privacy can include:⁴⁷
 - Locational and movement privacy (to protect against the tracking of spatial behaviour);
 - Communications privacy (to protect against the surveillance of conversations and correspondence); and
 - Transactional privacy (to protect against monitoring of queries/searches, purchases, and other exchanges).

Under this regulatory framework, **personally-identifiable information** (often referred to as PII), that is information that allows the identification of an individual, is accorded more protection than data representing environmental or other non-human phenomena.⁴⁸

⁴² 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/>.

⁴³ 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/>.

⁴⁴ The right to life, liberty and security of person as well as the right to be secure from unreasonable search and seizure. *Canadian Charter of Rights and Freedoms*, s 7-8, Part 1 of the *Constitution Act*, 1982, being Schedule B to the *Canada Act 1982* (UK), 1982, c 11.

⁴⁵ *R v Jarvis*, [2019] 1 SCR 488. <https://scc-csc.lexum.com/scc-csc/scc-csc/en/item/17515/index.do>

⁴⁶ *R v Jarvis*, [2019] 1 SCR 488. See also *R v Dyment*, [1988] 2 SCR 417, citing the control-based definition of Alan F Westin, *Privacy and Freedom* (New York: Atheneum, 1970). <https://scc-csc.lexum.com/scc-csc/scc-csc/en/item/375/index.do>

⁴⁷ Kitchin, “Getting Smarter About Smart Cities: Improving Data Privacy and Data Security,” 25.

⁴⁸ Information and Privacy Commissioner of Ontario, “Fact Sheet: What Is Personal Information?”

Privacy in a Digital Age

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

In the past, this information may have existed in physical form as a paper record and been stored in a single location. Virtually all PII nowadays is in digital form, which makes it much more efficient to manage through its lifecycle.

However, even non-personally identifiable information can be pieced together to build up an accurate profile of an individual – including their location, habits and preferences.⁴⁹ Whether this data processing is carried out by government or private sector actors, it has serious implications for privacy, as the Privacy Commissioner of Canada reflected on in the following quotation:⁵⁰

Privacy is nothing less than a prerequisite for freedom: the freedom to live and develop independently as a person, away from the watchful eye of a surveillance state or commercial enterprises, while still participating voluntarily and actively in the regular, day-to-day activities of a modern society such as socializing, reading the news, getting information about health issues or simply buying stuff.

The Legislative and Policy Context for Privacy

In Canada, privacy protections are set out in various pieces of legislation. 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.

Federal level:

- The Privacy Act - applies to information collected by the Government of Canada
- PIPEDA - applies to information collected for commercial purposes
 - PIPEDA underwent significant amendments with the passing of [The Digital Privacy Act](#) in June 2015 – specifically with regard to what constitutes valid consent when it comes to the use of individuals' information as well as providing for expanded enforcement powers for the Privacy Commissioner.

Provincial level:

Several provinces have enacted their own privacy legislation. Where this legislation has been deemed substantially similar to PIPEDA, the former may take precedence over the latter in certain cases.

⁴⁹ Keri Grieman, "Smart City Privacy in Canada," Report for the Office of the Privacy Commissioner of Canada (Samuelson-Glushko Canadian Internet Policy and Public Interest Clinic (CIPPIC), January 2019), https://smartcityprivacy.ca/wp-content/uploads/2019/03/Greiman-OPC-Report_Final-2019.pdf.

⁵⁰ 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/.

The following provincial privacy laws are considered equivalent in terms and protection to PIPEDA:⁵¹

- Alberta: [Personal Information Protection Act](#) overseen by the [Office of the Information and Privacy Commissioner of Alberta](#)
- B.C.: [Personal Information Protection Act](#) overseen by the [Office of the Information and Privacy Commissioner for British Columbia](#)
- Quebec: [Act Respecting the Protection of Personal Information in the Private Sector](#) overseen by the [Commission d'accès à l'information du Québec](#)

Municipalities in Canada are legally considered to be 'creatures of the province'; under Section 92 of the *Constitution Act, 1867*, provincial governments have legislative authority over the establishment of municipalities. As such, municipalities are generally subject to provincial legislation.

Some provinces have separate legislation setting out municipalities' obligations with respect to privacy protection:

- Ontario's [Municipal Freedom of Information and Protection of Privacy Act](#)

International

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 shaping norms around how software is developed and how personal information is protected within software.

- European Union General Data Protection Regulation ([GDPR](#)):
- California's [CCPA](#)

Addressing Privacy Risks Within OSS

- One potential strategy is to adopt additional tools to make sure the code is free from known vulnerabilities that could create a security risk and lead to a loss of privacy for data subjects.⁵²

⁵¹ Office of the Privacy Commissioner of Canada, "Provincial Laws That May Apply Instead of PIPEDA," May 29, 2017, https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipeda/r_o_p/prov-pipeda/.

⁵² Nicole Olsen, "Open Source Projects and the GDPR," Privacy Policies, accessed September 2, 2022, <https://www.privacypolicies.com/blog/gdpr-open-source-projects/>.

Section 3: Assessing and Mitigating Security and Privacy Risks

Protecting Your Organization from OSS Security Risks

An IT security framework can mitigate various risks stemming from OSS use in your organization. Such a framework should include the following components:⁵³

- **Establishing an organizational IT security policy.** This policy should outline how employees work and collaborate on corporate networks while respecting the organization's position on managing security risks. Be sure to review and update this policy regularly so that it reflects the current threat environment.
- **Allocating appropriate levels of access** - Define the roles and responsibilities of system administrators, and limit the number of individuals with this level of access.
- **Developing system administration procedures** that define how system administrators should securely manage and maintain your organization's networks and software.

Further Resources

⁵³ Canadian Centre for Cyber Security, "Security Considerations When Using Open Source Software (ITSAP.10.059)."

Module 5: Accessibility & Maintenance of Open Source Software

MODULE TO BE DEVELOPED

Learning Objectives

In this module we will:

- Discuss key sustainability issues related to OSS
- Outline common challenges that organizations face in maintaining their OSS solutions
- Identify key elements of successful stewardship of OSS solutions

Introduction

Key Takeaways

Further Resources

Module 6: Working with Open Source Software Across Jurisdictions

MODULE TO BE DEVELOPED

Learning Objectives

In this module we will:

- Review the interjurisdictional issues raised in previous modules
- Discuss key considerations when forming multi-jurisdictional teams around a particular piece of software
- Discuss resource sharing between different levels of government
- Explore what it means to participate in a global community of codebase stewardship

Introduction

Key Takeaways

Further Resources

[Intergovernmental Software Collaborative](#)

References

[INSERT FINAL BIBLIOGRAPHY HERE]

DRAFT

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 . (Source: Atlassian . See also the Agile Manifesto)
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)
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
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)
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
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
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
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
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)
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)
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)

Product Owner	<p>"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)</p>
Pull request	<p>"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</p> <p>See also: Distributed version control</p>
Repository	<p>"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</p> <p>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.</p> <p>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."</p> <p>See also: Distributed version control</p>
Request for Information (RFI) / Sources Sought Notice (SSN)	<p>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)</p>
Request for Proposals (RFP)	<p>"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</p>

	and conditions, timeliness and activities involved, and what respondents should include in their RFP response.” (Source: Gartner)
Request for Quote	“A solicitation method used to obtain price, cost, delivery, and related information from suppliers.” (Source: US Government Services Administration)
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)
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.).

DRAFT