Survey of CPI Production Systems

Collin Brown and Steve Martin

2025-02-03

Table of contents

Preface

We would like to acknowledge the helpful feedback and numerous contributions from our colleagues in the Workstream on CPI Systems Architecture in the Task Team on Scanner Data under the UN Committee of Experts on Big Data and Data Science for Official Statistics.

Survey Methodology Notes

We follow a few standard practices for reporting on survey data.

Value Suppression

To preserve anonymity, we do not disclose any values if there are 2 or fewer respondents that take on the value. As a result, throughout the report, certain tables and figures may be presented in a way where certain categories are omitted or grouped together.

1 Introduction

We created this survey about the state of Consumer Price Index (CPI) Production Systems on behalf of the Task Team on Scanner Data under the UN Committee of Experts on Big Data and Data Science for Official Statistics.

While this report is specific to the state of CPI Production Systems at National Statistics Organizations (NSOs) around the world, our hope is that some of the content in this report is also useful for a wider audience maintaining similar kinds of systems. As such, we attempt to explain our results in a general way and highlight opportunities where our survey approach and findings could be applied in related settings.

1.1 Motivation

In our time working at National Statistics Organizations (NSOs), we have encountered some extremely complicated systems that exist in order to produce various analytical and data products such as consumer price indexes, national accounts figures, or labour force statistics. These complicated systems and the teams who maintain them are the subjects of this survey and write up. To reduce ambiguity, we refer to these systems as **Complex Analytical Systems** throughout this report.

Complex Analytical Systems involve significant amounts of code, documentation, and other non-code artifacts such as Excel Workbooks that carry out complex business logic in order to transform input data into output data. Additionally, they are often developed entirely or in large part by people with backgrounds in Economics, Statistics, Mathematics, or another area related to the domain of Official Statistics.

These Complex Analytical Systems differ from traditional software systems in a number of important aspects[1]:

[1] For readers who are unfamiliar with the Consumer Prices business domain, these are techniques that are used to calculate period-over-period price changes from a given data source.

Complex Analytical Systems	Typical Software System
Multiple distinct scripts that are run sequentially and perform complex data manipulations.	One code base representing an entire application.
Running time measured in minutes/hours	Running time measured in milliseconds.
Human in the loop activities to interpret results.	Completely autonomous system.
Ad-hoc (messy) data gathered from whatever data sources are available.	Highly structured data whose schema is designed in lock step with the rest of the system.
Batch workloads that are run manually (or semi-manually).	System running continuously in an event loop waiting for user input.
Operate on a large fraction of an entire table quickly.	Search for one specific record in a large table quickly.

Due to differences like those mentioned above, there is not a perfect mapping between best practices from the software engineering world and pain points currently experienced by teams maintaining Complex Analytical Systems. However, there are certainly some best practices from software engineering that are highly appropriate to solve some of the problems faced in the development and maintenance of Complex Analytical Systems.

To this end, we hope our survey can help bridge the gap between well-understood industry best practices from the world of software engineering, and those aspects of Complex Analytical Systems that could benefit from these best practices. Our hope is that the insights gained and the survey methodology deployed may be valuable for other Complex Analytical Systems facing similar challenges.

1.2 Why Did We Run This Survey?

In our experience, we've noticed that many teams who are responsible for Complex Analytical Systems struggle with managing many aspects of system complexity.

Complex Analytical System business domain teams are typically comprised of individuals with strong analytical skills and significant domain knowledge, however, they often do not have specific training in software engineering concepts. Therefore, they are often not exposed to the significant body of knowledge that has been developed over decades to deal with the kinds of system complexity problems that software developers are routinely exposed to.

We have also found that individuals in these business domains are often missing the vocabulary and concepts to articulate the state of their Complex Analytical Systems. As a result, when

these individuals try to explain where they are struggling to a more IT-oriented audience, miscommunication often results, and it becomes difficult to arrive at reasonable solutions.

In this survey, we ask questions that capture several germane aspects of system organization, team organization, technology choices, and business outcomes using language, terms, and conceptual models that are more familiar to individuals on these business domain teams. Our rationale for doing this is threefold.

1. Measure and describe the state of many Complex Analytical Systems around the world within a specific business domain (CPI Production Systems).
2. Provide some concrete and practical suggestions to address common areas of struggle within this domain across many NSOs.
3. Expose people from these business domain teams to software engineering concepts that are relevant in the development and maintenance of Complex Analytical Systems.

While this report is tailored towards a Consumer Prices domain audience, we welcome and encourage readers from different domains to read through this report. We make significant efforts to avoid using too much domain-specific jargon, and present findings in a way that should comprehensible to a more general audience. In Chapter 10, we elaborate on aspects of our survey we believe to have high external validity, provide some practical suggestions that are applicable to Complex Analytical Systems in general, and describe some productive areas of future exploration that are not limited to the Consumer Prices business domain.

1.3 Overview of CPI Production Systems

With the above motivation in mind, we conduct this survey for CPI Production Systems specifically, which are a kind of Complex Analytical System described in Section 1.1. More precisely, these systems take data on the price of consumer goods purchased throughout an economy and calculate period-over-period price changes of these goods. These price changes are ultimately mapped to a taxonomy of product categories, with the highest level of the taxonomy being the monthly "all items" CPI that is commonly used when discussing the overall level of inflation.

The recent adoption of alternative data sources[2] in the calculation of CPIs has further increased the complexity of these systems, and has increased the importance of skills in newly emerging disciplines such as Data Science, Data Engineering, and Analytics Engineering.

[2]Note that there is a small sample size caveat with observations involving Other Mix teams. Had the sample size been larger, the observations we observed with this team type may not have been as extreme as what we observed in the survey.

1.4 Related Work

We borrow and adapt several ideas presented in Skelton, Pais, and Malan (2019) such as the concepts of Stream-aligned teams and the Flow of change in our survey. These concepts (discussed in greater detail in the following sections) can be applied to understand how teams are organized around the various steps in a complex data processing workflow.

We also borrow a number of ideas from Forsgren, Humble, and Kim (2018). Particularly, multiple of the DevOps Research and Assessment (DORA) metrics they present are highly relevant in measuring the business outcomes of teams that maintain CPI systems.[3]

We believe that multiple ideas presented in Dehghani (2022) are highly applicable to the CPI Systems under study. Specifically, we believe that the concept of Data-as-a-Product (and teams organized around these Data Products) provides a useful framework for thinking about how these systems and teams interact with one another. We also contrast domain-oriented decentralized teams with centralized teams.

Although we do not make specific references to it in our survey, we believe the Reproducible Analytical Pipeline (RAP) work by NHS (2017) does a good job at explaining how teams can introduce relevant tools and practices to workloads oriented around data processing.

Throughout this write up, we distinguish between software systems and IT professionals being **embedded in domain teams** versus being centralized **outside of domain teams**. Organizing software system architecture around business domains is not a new idea in software engineering (see Domain Driven Design by Evans (2004), and more recently Software Architecture: The Hard Parts by Ford et al. (2021) and Fundamentals of Software Architecture by Richards and Ford (2020), for example). However, we believe this distinction may not be well understood or formalized in the context of the teams who maintain the kinds of Complex Analytical Systems described in Section 1.2, so we pay special attention to this distinction throughout this write up.

1.5 How This Report Is Organized

This report is presented in the order that the survey was conducted, with findings presented along the way.

- Chapter 2 covers the key conceptual models and terminology used to articulate concepts about system and team organization.
- Chapter 3 analyzes our findings with respect to system and team organization.

[3]We suspect that this is slightly biased by the fact that the majority of Stream-Aligned teams in our sample are comprised of domain-analysts only rather than having both domain-embedded analysts **and** domain-embedded IT professionals.

- Chapter 4 covers some high-level questions on the use of tools and technologies required to develop and maintain CPI Production Systems.
- Chapter 5 covers questions about the age and update frequency of systems.
- Chapter 6 covers questions about the number of individuals required to participate in system changes.
- Chapter 7 covers questions about a concept called lead time, which measures the end-to-end time required to implement a change to a software component.
- Chapter 8 covers questions about the usage of alternative data in CPI Production Systems.
- Chapter 9 covers the challenges CPI Production System teams face with respect to maintaining their systems.
- Chapter 10 concludes with a summary of the most notable findings from the survey, some practical insights to address some common areas of struggle, and some areas of future work.

1.6 Note on Confidentiality and Privacy

As part of the administration of this survey, we ensured respondents that their data will be treated confidentiality. Therefore, no individual response data are made available in this report; all results presented are aggregated over all respondents.

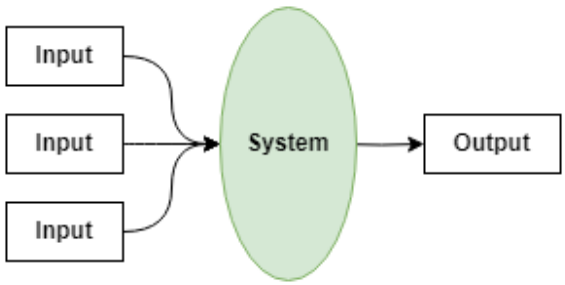
2 Concepts

This survey is concerned with the interaction between teams, software systems, and the flow of change from input (upstream) data to output (downstream) data.

Since this survey is specifically focused on CPI Production Systems, we orient the flow of change around key Generic Statistical Business Process Model (GSBPM) steps. Some of these steps are quite generic and common to most data teams (e.g., Ingestion, Processing), while others are very specific to CPI teams (e.g., Elementary Index Calculation, Elementary Index Aggregation).

The concepts introduced in the rest of this page can easily be applied to other domain-specific workflows, but everything that follows is implicitly explained in the context of these GSBPM steps.

2.1 Systems and Teams



(a) System Diagram

For the purposes of this survey, we define a **system** as any indivisible (atomic) software component that takes one or more data inputs and produces one or more data outputs.

When we refer to an indivisible software component, we mean that the component runs "entirely or not at all" with respect to the transformation of inputs into outputs. For example, if there is one Python script that reads a file and writes an intermediate file, and a second R script that reads the intermediate file and produces another output file, we would consider this to be two separate systems.

The types of systems developed and maintained by the teams being surveyed can vary wildly. Therefore, we intentionally keep the definition of system vague so that it captures the key

activity of transforming data without imposing any assumptions about how data are transformed.

A **team** is defined as a group of individuals who maintain one or more systems. Teams can be composed of many different types of professionals, but for the purpose of this survey we distinguish between those who are Information Technology (IT) professionals (e.g., software developers) and those who are non-IT analysts (e.g., economists, statisticians)[1].

Importantly, we also distinguish whether these teams are embedded in the price-statistics domain of the organization or work elsewhere in the organization. The detail we care about here is whether the team developing and maintaining one or more systems shares business context and domain knowledge with the team making use of these systems, or whether they do not share this business context and domain knowledge.

In total, we define the following 5 team types for the purposes of this survey.

Team Type	Description
Corporate IT	Information Technology (IT) professionals who are part of an organization-wide central group (i.e., **not functionally embedded** in the price statistics team).
Domain-Embedded IT	IT professionals who **are functionally embedded** in the price statistics team (i.e., part of the price-statistics team, not the corporate IT department).
Domain-Embedded Analysts	Professionals without a formal IT background (e.g., economists, statisticians) who **are functionally embedded** in the price statistics team.
Analysts Elsewhere in the Organization	Professionals without a formal IT background who are **not functionally embedded** in the price statistics team.
External Consultants or Contractors	Professionals outside of the organization to whom system development/maintenance work is contracted.

[1]Note that there is a small sample size caveat with observations involving Other Mix teams. Had the sample size been larger, the observations we observed with this team type may not have been as extreme as what we observed in the survey.

2.2 Flow of Change

The business processes that NSOs follow to create their country's CPI is a special case of the General Statistical Business Process Model (GSBPM). We base our flow of change on the GSBPM model and specialize it slightly towards the CPI domain. We define each step in our flow of change as follows.

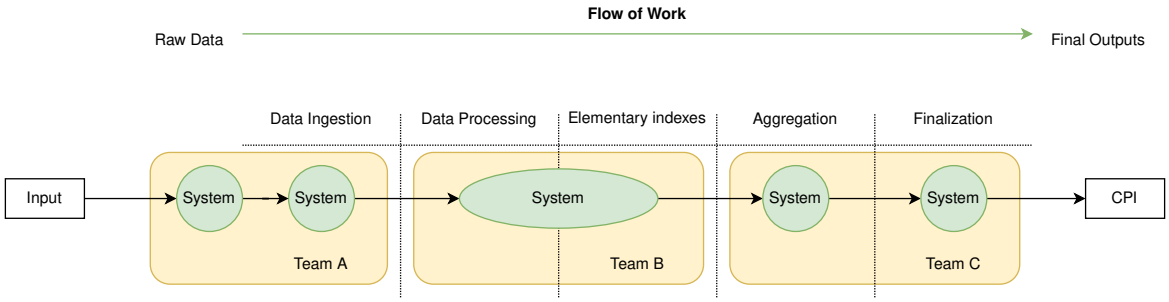


Figure 2.2: Flow of Work

Process	Explanation
Data Ingestion	Activities to bring acquired data into a machine-readable state where it is ready for further processing. For example, entering paper surveys into an electronic database or querying a REST API to collect prices from a website.
Data Processing	Activities to clean, validate, correct, impute, or otherwise adjust the data so that it is in a state where it is ready to be used to produce elementary indexes.
Elementary Indexes	Calculate price indexes from the processed data for a given geography and product category (i.e., elementary aggregate) at a point in time[2].
Aggregation	Aggregate the elementary price indexes into higher-level indexes (e.g., "All Items" CPI)[3].
Finalization	Store price indexes for subsequent use as part of analytical activities, and eventual dissemination.

[2]We suspect that this is slightly biased by the fact that the majority of Stream-Aligned teams in our sample are comprised of domain-analysts only rather than having both domain-embedded analysts **and** domain-embedded IT professionals.

[3]For readers who are unfamiliar with the Consumer Prices business domain, these are techniques that are used to calculate period-over-period price changes from a given data source.

Figure 2.2 overlays all of the concepts discussed so far onto a single diagram. The flow of work (raw data to final CPI) can be read from left to right, with the green ovals representing systems and the boxes drawn around one or more systems representing the teams that own them.

The systems in these diagrams can be "piped" together, implying that the output of one system becomes the input to the next system. To simplify the notation, the intermediate input/output boxes are simply shown as an arrow from one system node to the next system node.

In the event that multiple teams collectively maintain a very large system, a team can be interpreted as the larger organizational unit that oversees the various sub-teams involved.

2.3 Putting it All Together

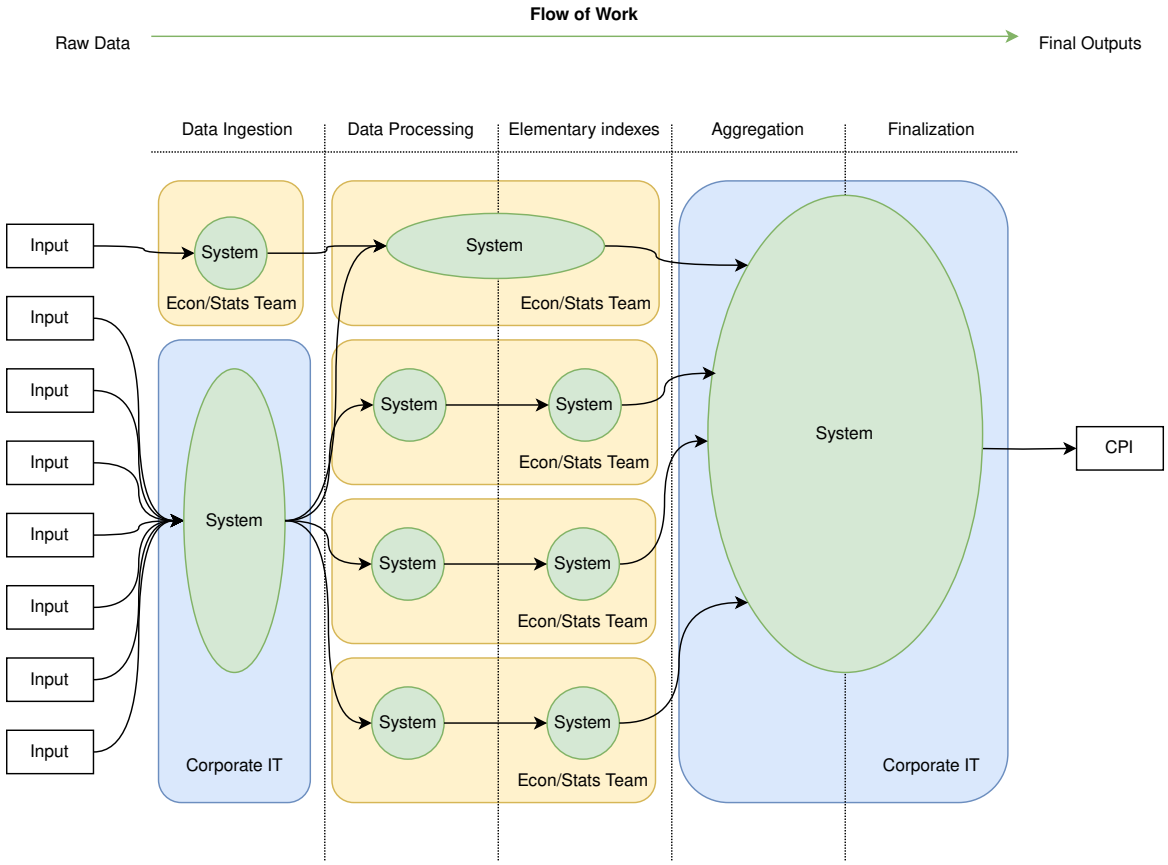


Figure 2.3: Putting it All Together

Figure 2.3 shows an example of what one of these diagrams could look like for a real NSO. A description of team/system organization that could correspond to this example is described below.

- A **corporate IT** team maintains a large Java application with an application-specific database that handles ingestion of most data sources[4].
- A team of **domain-embedded analysts** (e.g., economists and statisticians) maintain a system written in Python, using pandas to ingest data from one source that does not integrate with the corporate system[5].
- A team of domain-embedded analysts maintain a series of systems, using some combination of R and Python, that perform data processing and computation of elementary indexes.
  - The top system handles both the data processing step **and** the elementary index calculation step, implying it would not be straightforward to decouple these two concerns.
  - The bottom three cases have two distinct systems that handle data processing and elementary index calculation separately.
- A second system owned by corporate IT aggregates the elementary indexes and stores the resulting price indexes in a database.

2.4 Modular vs. Monolithic Systems

We do not impose any restrictions on the number of systems that can belong under a team or a GSBPM step. Moreover, we do not impose any restriction on how many GSBPM steps a system can span. This flexibility lets us express the extent to which systems span across these GSBPM steps.

Figure 2.4 and Figure 2.5 show contrasting examples of two extreme scenarios. Figure 2.4 is an example where there is one system owned by one team that handles all 5 GSBPM steps for all inputs. In contrast, Figure 2.5 shows the scenario where each GSBPM step has a dedicated system that does not cross between GSBPM steps.

It is unlikely that any NSO will have systems organized like one of these extremes, rather most organizations will likely fall somewhere in between.

[4]We are skeptical that these two facts can be true at the same time. Our suspicion is that some respondents have not explicitly thought about version control as a distinct problem with purpose-built tooling that exists to solve it.

[5]In general, the Consumer Price Index is a non-revisable product, meaning that it is not straightforward to "roll back" a change in the same way that would be possible in other settings. Because of this, there is a certain level of due-diligence required for large system changes, meaning there are some limits on how frequently CPI Production Systems can be updated.

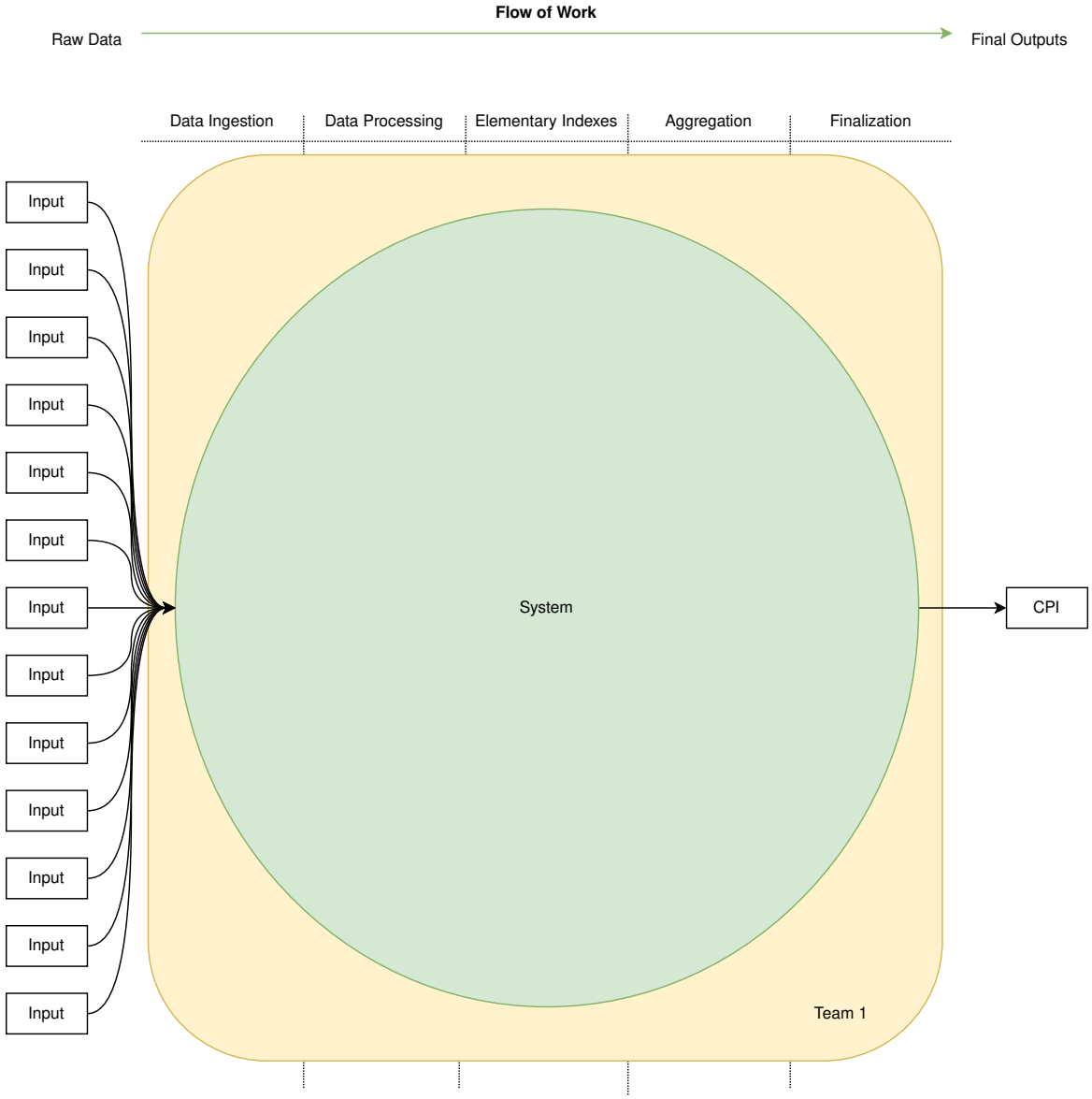


Figure 2.4: Perfect Monolith Example

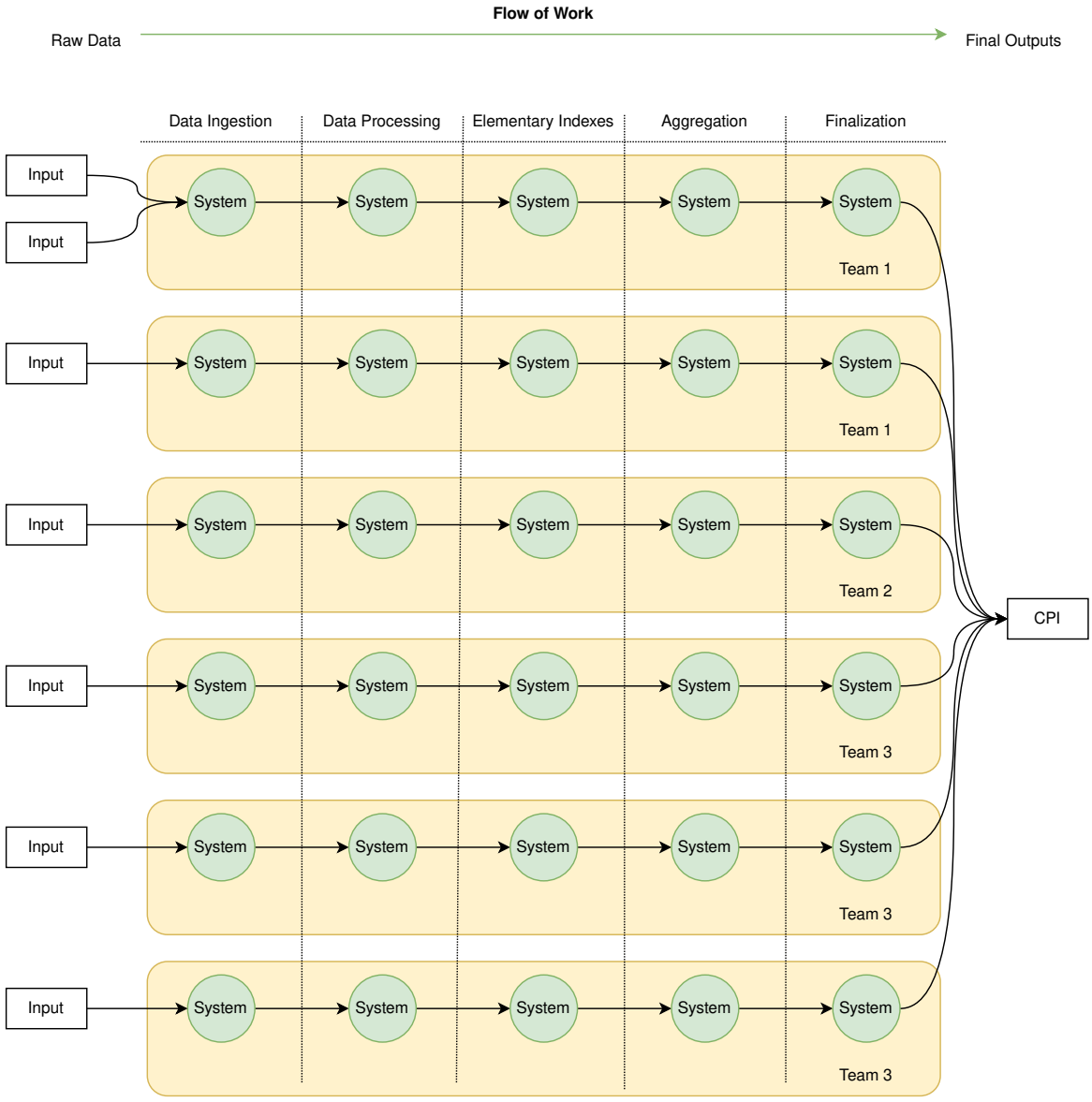


Figure 2.5: Perfect Modular Example

2.5 Representative Groups of Systems

To get a complete understanding of each NSO's team organization and system topology, we would need to have a comprehensive whiteboarding session with representatives from each NSO to fully articulate their CPI team structures and system architectures. This is obviously not feasible.

To get around this impracticality, we introduce the concept of a **representative system group** to the survey, which lets respondents describe one or more groups of systems that follow a typical pattern in their organization. Figure 2.6 shows what one such system group might look like using the example shown earlier in Figure 2.3.

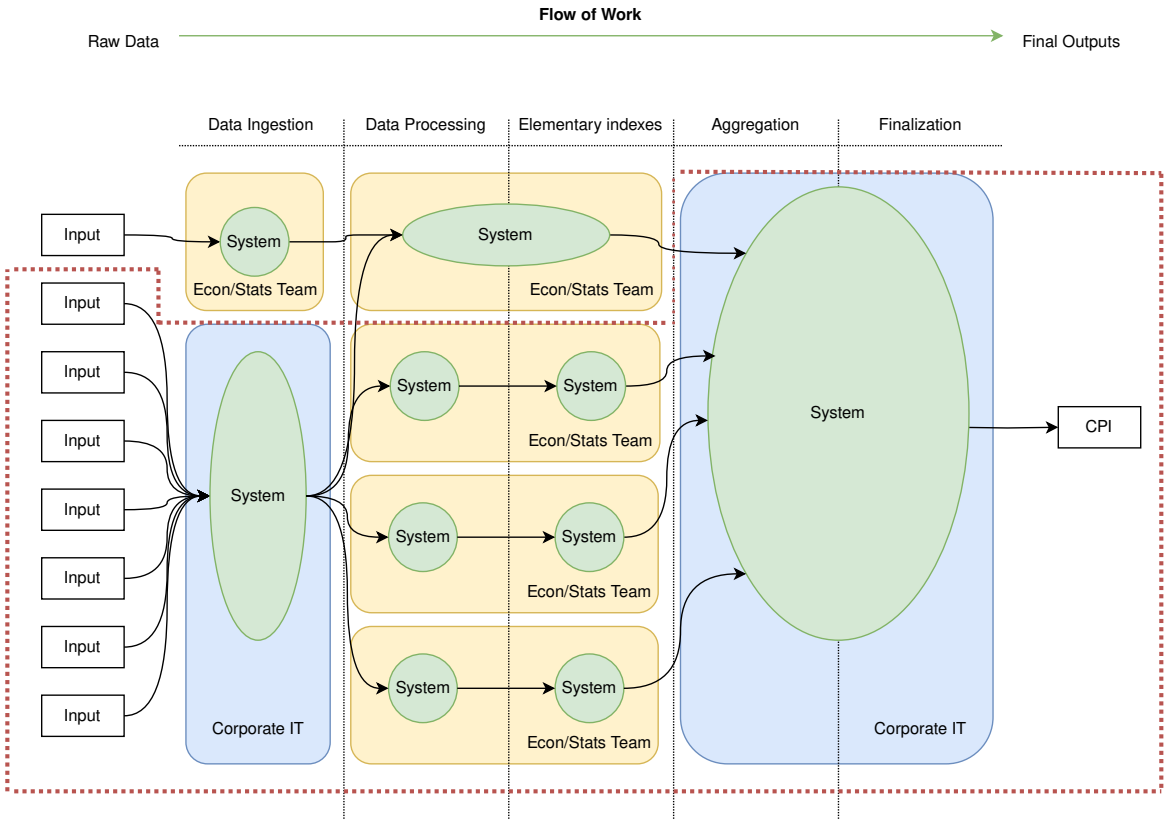


Figure 2.6: Representative System Group

The systems enclosed by the dashed red line in the diagram constitute a representative group, composed of three subgroups of systems.

- There is a single system for data ingestion that is maintained by the organization's corporate IT department (system group 1).

- There are three systems that handle data processing and three systems that handle elementary index calculation. Each pair of systems is maintained by one or more small teams of domain-embedded analysts (system group 2).
- There is one large system that handles aggregation and finalization of elementary index calculations. This system is also maintained by the corporate IT department (system group 3).

Our rationale behind asking respondents to describe their representative system group is threefold:

1. Significantly reduce the response burden compared to completely articulating the state of **all** systems and teams that produce the CPI.
2. Reliably capture "system boundary" points where there is a switching between one group of systems and the next. While we do not have the resolution to know which GSBPM steps are crossed **within** a representative system group, we know for certain that there is a system boundary **between** two distinct representative system groups.
3. Reliably capture which kinds of teams occur together.

3 Systems and Teams Analysis

Our first objective in this analysis is to describe the kinds of system and team organizations we encountered in our survey. We start with an explanation of our methodology, followed by some notable findings.

3.1 Steps Where Systems are Coupled

One important piece of system architecture information we were interested in is cases where systems span more than one GSBPM step along the flow of change. Using our definition of "system" from Chapter 2, a system that spans two or more GSBPM steps implies that it would not be straightforward to modify just one GSBPM step in the system without affecting the others[1]. This relates to the concept of coupling in software engineering, which measures the extent to which distinct software modules depend on each other.

[1]For example, a single Python script that processes data using Pandas and immediately performs a price index calculation on the processed data frame while it's still in memory is a system that spans the (1) Processing and the (2) Elementary Indexes GSBPM steps. Such a system has coupled the Processing logic and the Elementary Indexes logic because it is not straightforward to change just the Processing component without also needing to consider how the Elementary Indexes component is affected.

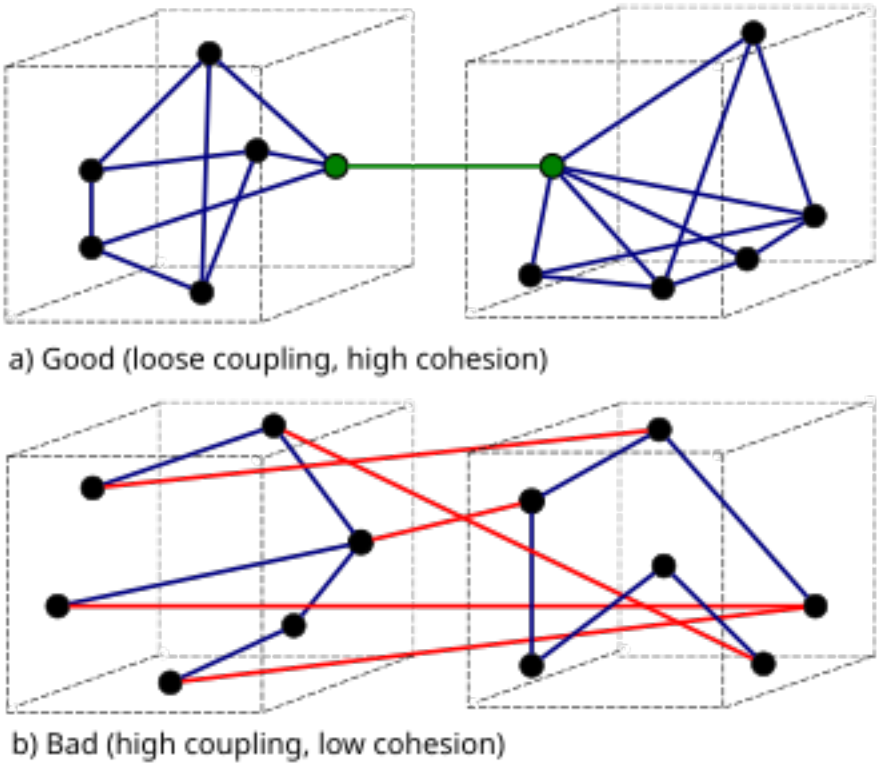


Figure 3.1: Conceptual diagram of loose and high coupling. Source: https://en.wikipedia.org/wiki/Coupling_(computer_programming)

We asked the following question to get a coarse-grained understanding of systems that spanned more than one GSBPM step.

Indicate which steps are handled by the same system, for up to five typical systems.

For example, a setup with one system that handled "Data ingestion", "Data processing", and "Elementary indexes", a second system that handled "Data processing" and "Elementary indexes", and a third system that handled "Elementary indexes", "Aggregation", and "Finalization", would look like this.

	Data ingestion	Data processing	Elementary indexes	Aggregation	Finalization
System 1	☑	☑	☑		
System 2		☑	☑		
System 3			☑	☑	☑

	Data ingestion	Data processing	Elementary indexes	Aggregation	Finalization
System 1	☐	☐	☐	☐	☐
System 2	☐	☐	☐	☐	☐
System 3	☐	☐	☐	☐	☐
System 4	☐	☐	☐	☐	☐
System 5	☐	☐	☐	☐	☐

Figure 3.2: Systems Spanning Multiple GSBPM Steps Question

We note how frequently systems cross certain GSBPM steps in Table 3.1.

Table 3.1: Systems that span across multiple GSBPM steps

Data ingestion	Data processing	Elementary indexes	Aggregation	Finalization	Frequency
X	**X**				21
	X	**X**			6
		X	**X**		0
			X	**X**	6
X	**X**	**X**			12
	X	**X**	**X**		8
		X	**X**	**X**	11
	X	**X**	**X**	**X**	4
X	**X**	**X**	**X**	**X**	20

The most common pattern we observed was that the same system handled both data ingestion and data processing with 21 occurrences. We suspect this is is fairly common as the two activities are conceptually similar, and often leverage similar tooling. For example, if an analyst wrote a web-scraping script to collect price data from a certain website, it's reasonable that they would also perform some data cleaning activities in the same script before writing the output to persistent storage.

The second most common pattern we observed was the "Monolith" pattern, where we observe one system spanning all 5 GSBPM steps 20 times. We elaborate on this concept later in this section, but our suspicion is that systems like these mostly fall into one of two categories: (1) a small team and an Excel Workbook that handles everything from ingestion to finalization or (2) a single system managed by an IT team.

Monolithic systems are not inherently bad. If they are carefully designed for maximum cohesion and minimal coupling, they can be maintainable. However, if monoliths **emerge accidentally**, they often involve many tightly coupled components and become very difficult to maintain in the long run.

The next two most frequent coupling patterns are **ingestion-processing-elementary indexes** (12 occurrences) and **elementary indexes-aggregation-finalization** (11 occurrences). The latter makes some sense to us as these three concerns can be quite bespoke and tailored to the price statistics business domain. The former, however, surprised us as ingestion and processing are fairly generic data preparation activities, while elementary index calculation is a highly specific price statistics activity.

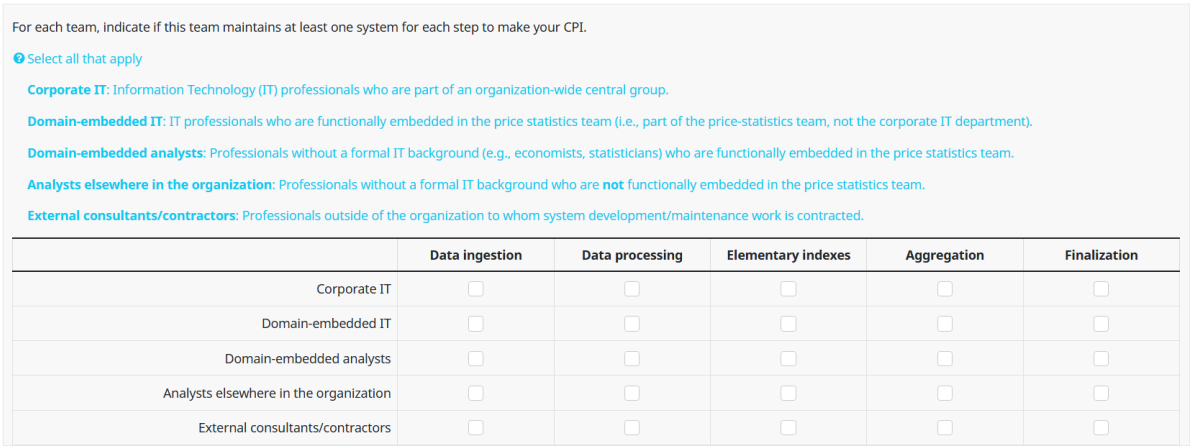
We were also surprised to learn that there are zero occurrences of a system spanning **elementary indexes-aggregation**. In the price statistics business domain, these concepts are often

closely related, so we thought there would be more cases where a single system handled both steps[2].

3.2 Team Types at Each Step

The next concept we try to learn in the survey is which types of teams discussed in Chapter 2 are present at each GSBPM step in the flow of change.

We present respondents with the following question.


For each team, indicate if this team maintains at least one system for each step to make your CPI.

Select all that apply

Corporate IT: Information Technology (IT) professionals who are part of an organization-wide central group.

Domain-embedded IT: IT professionals who are functionally embedded in the price statistics team (i.e., part of the price-statistics team, not the corporate IT department).

Domain-embedded analysts: Professionals without a formal IT background (e.g., economists, statisticians) who are functionally embedded in the price statistics team.

Analysts elsewhere in the organization: Professionals without a formal IT background who are **not** functionally embedded in the price statistics team.

External consultants/contractors: Professionals outside of the organization to whom system development/maintenance work is contracted.

	Data ingestion	Data processing	Elementary indexes	Aggregation	Finalization
Corporate IT	☐	☐	☐	☐	☐
Domain-embedded IT	☐	☐	☐	☐	☐
Domain-embedded analysts	☐	☐	☐	☐	☐
Analysts elsewhere in the organization	☐	☐	☐	☐	☐
External consultants/contractors	☐	☐	☐	☐	☐

Figure 3.3: Systems Spanning Multiple GSBPM Steps Question

Table 3.2: Common Team Combinations **within** GSBPM Steps

Corporate IT	Domain IT	Domain Analysts	Other Analysts	Consultants	Team Type	Frequency
		X			Domain Analyst Only	120
	X	**X**			Domain Analysts and Domain IT	25

[2]An example of a system that performs both elementary index calculation and aggregation would be if the same script calculated component price changes in apples, bananas, and oranges, then the same script also calculated the overall price change in fruit from these component price changes.

Corporate IT	Domain IT	Domain Analysts	Other Analysts	Consultants	Team Type	Frequency
X					Corporate IT-Only	33
	X				Domain IT-Only	20
X	**X**				IT-Only	11
X		**X**			Corporate IT and Domain Analysts Only	45

Table 3.2 shows how frequently certain team combinations were reported **within** any given GSBPM step.

We were surprised to see that Corporate IT and Domain Analyst teams occurred almost twice as frequently as teams with Domain-embedded IT **and** Domain-embedded analysts. In particular, this suggests that technical expertise offered by IT personnel is often centralized outside of the domain area rather than functionally embedding IT personnel within the domain team.

3.3 Capturing System And Team Organizations Together

3.4 Commonly Occurring System and Team Topologies

We begin by sharing some commonly occurring system and team topologies that occurred in our responses.

We perform k-means clustering with a cluster size of 3 on each respondent's answer the "system topology" part of the Representative System Group question[3].

We were not able to find any rule or explanation that perfectly split respondents into some number of clusters for this question.

However, we were able to identify some high-level patterns that were consistent across the 3 groups. We use these patterns to characterize 3 archetypes that are explained below along with some illustrative examples from each cluster.

[3]When working with this quantity of data, it is important to pay attention to the data types of each column and to choose the most parsimonious data types. For example, if 3 columns can be correctly represented with boolean, 16-bit integer, and 32-bit integer data types, there are significant memory savings to be gained by casting the variables to these types upfront.

3.4.1 One or more System Groups Span the Entire Flow of Change

	Ingest	Process	Elementals	Aggregate	Finalize
System Group 1	TRUE	TRUE	TRUE	TRUE	TRUE
System Group 2	FALSE	FALSE	FALSE	FALSE	FALSE
System Group 3	FALSE	FALSE	FALSE	FALSE	FALSE
System Group 4	FALSE	FALSE	FALSE	FALSE	FALSE
System Group 5	FALSE	FALSE	FALSE	FALSE	FALSE

3.4.2 There is exactly one "split point" between System Groups

	Ingest	Process	Elementals	Aggregate	Finalize
System Group 1	TRUE	TRUE	FALSE	FALSE	FALSE
System Group 2	FALSE	FALSE	TRUE	TRUE	TRUE
System Group 3	FALSE	FALSE	FALSE	FALSE	FALSE
System Group 4	FALSE	FALSE	FALSE	FALSE	FALSE
System Group 5	FALSE	FALSE	FALSE	FALSE	FALSE

3.4.3 There are two or more split points between system groups

	Ingest	Process	Elementals	Aggregate	Finalize
System Group 1	TRUE	FALSE	FALSE	FALSE	FALSE
System Group 2	FALSE	TRUE	FALSE	FALSE	FALSE
System Group 3	FALSE	FALSE	TRUE	TRUE	TRUE
System Group 4	FALSE	FALSE	FALSE	FALSE	FALSE
System Group 5	FALSE	FALSE	FALSE	FALSE	FALSE

3.5 Assigning Architectures to Organizations

We use responses to the Representative System Group question to assign NSOs to one of the three architecture types defined below.

3.5.1 Monolithic Architecture

We assume that an NSO has a monolithic architecture if any of their system groups span all 5 GSBPM steps[4],[5].

Our reasoning for this assumption is that if the respondent thought a system group spanning all 5 GSBPM steps was similar enough that there was no need to split any subset of it into a second system group, it is probably due to coupling between one or more systems in the system group.

It is important to note that since we do not ask questions about the quantity or span of systems, we cannot know for certain whether or not a system group spanning all 5 GSBPM steps is truly monolithic[6].

For example, a system group that spans all 5 GSBPM steps could be comprised of one system that truly is monolithic with respect to the 5 GSBPM steps, or it could be comprised of 5 or more distinct systems that the respondent felt were sufficiently representative of their typical workflow[7].

Nevertheless, we needed to make some simplifying assumptions to make the response burden of this survey realistic, and we do not believe this assumption is too far fetched.

3.5.2 Hybrid Architectures

We classify any NSO that (1) is not a monolith and (2) has exactly two system groups with any split between them as having a Hybrid architecture type.

Our rationale for this category is to look at cases where there is one "boundary point" between two distinct system groups.

[4]Note that there is a small sample size caveat with observations involving Other Mix teams. Had the sample size been larger, the observations we observed with this team type may not have been as extreme as what we observed in the survey.

[5]We are skeptical that these two facts can be true at the same time. Our suspicion is that some respondents have not explicitly thought about version control as a distinct problem with purpose-built tooling that exists to solve it.

[6]We suspect that this is slightly biased by the fact that the majority of Stream-Aligned teams in our sample are comprised of domain-analysts only rather than having both domain-embedded analysts **and** domain-embedded IT professionals.

[7]For readers who are unfamiliar with the Consumer Prices business domain, these are techniques that are used to calculate period-over-period price changes from a given data source.

3.5.3 Modular Architectures

We classify any NSO that (1) is not a monolith, (2) is not a hybrid, and (3) has three or more system groups with any splits between them as having a Modular architecture type.

While this kind of architecture is not necessarily perfectly modular, there are definitely two or more "boundary points" between three or more distinct systems.

There are a couple of important points to note about Hybrid and Modular architectures as defined above.

1. Unlike Monolithic architectures where we have to make an assumption about the internal structure of a system group, here we know for certain that there are explicit "boundary points" between two system groups that the respondent deemed sufficiently different to not be grouped together.
2. Using our definition of Systems given in Chapter 2, Hybrid and Modular architectures are guaranteed to pass output data from one system as input data to another system at least once[8].

3.6 Assigning Team Types to Organizations

We assign each organization one or more of the following team types using the following definitions.

- **Stream Aligned team**: A system group is maintained by domain-analysts, domain-IT, or both.
- **IT-Only team**: A system group is maintained by corporate-IT, domain-IT, or both.
- **Analyst-Only team**: A system group is maintained by domain-analysts, elsewhere-analysts, or both.
- **Other Mix team**: Assigned to organizations that have not been assigned to any of the above three team types.

Note: Unlike the architecture definitions earlier in this section, these team types are **not mutually exclusive** in the way we define them.

[8] In general, the Consumer Price Index is a non-revisable product, meaning that it is not straightforward to "roll back" a change in the same way that would be possible in other settings. Because of this, there is a certain level of due-diligence required for large system changes, meaning there are some limits on how frequently CPI Production Systems can be updated.

For example, an organization could have one system group maintained by a Stream Aligned team, and a second system group maintained by an IT-Only team[9].

3.7 Are Certain Team Types Correlated with Certain Architectures?

We define similarity between organizations having a particular team type **and** a representative system with a particular architecture as follows.

$$\text{Similarity} = \frac{\sum_{\text{respondents}} \text{Has Architecture Type} == 1 \wedge \text{Has Team Type} == 1}{\sum_{\text{respondents}} \text{Has Team Type} == 1}$$

The rationale for this similarity metric is that we care about what fraction of organizations with a particular team type also uses a given architecture type relative to how many instances of that team type are observed in the sample. Note that this can also be interpreted as a conditional probability where **team type** is the conditioning variable.

	Monolith	Hybrid	Modular
Stream Aligned Team	0.465116279069767	0.13953488372093	0.395348837209302
IT-Only Team	0.346153846153846	0.153846153846154	0.5
Analyst-Only Team	0.4	0.2	0.4
Other Mix	> 0.75	< 0.25	< 0.25
Sample Average	0.491803278688525	0.147540983606557	0.360655737704918

There are a few notable observations to point out in the above distribution[10].

1. IT-Only teams are more likely than average to be present in NSOs with Modular representative systems.
2. Stream-Aligned teams are more likely to be present in NSOs with Monolithic representative systems compared to IT-Only teams.
3. Other Mix teams were **much** more likely to be present in NSOs with Monolithic representative systems compared to all other team types[11].

Throughout the remainder of this report, we look at how various practices and outcomes are associated with the presence of each architecture type and team type.

[9] For example, if there is evidence that certain team structures, team interaction modes, and system architectures are more effective than others, then organizations that adopt the improved team topologies and system architectures may see reduced maintenance costs and faster delivery times, among other benefits.

[10] `UPSERT` is a portmanteau of the common `INSERT` and `UPDATE` database operations.

[11] It is important to note that a relatively small fraction of the sample contained Other Mix teams, so this result could be partly or entirely explained by a small number of outliers.

4 Tools and Technologies

In this section, we look at various tools and technologies in use by NSOs.

4.1 Version Control System Usage

Complex Analytical Systems, such as the systems used in monthly CPI Production, often need to synchronize and integrate multiple versions of related code, data, and documentation. This is especially important for creating reproducible analytical pipelines (NHS (2017)), or enabling multiple individuals to work concurrently on the same project.

Version Control Systems (VCS) are tools that systematically manage changes in a codebase over time. At the time of writing, Git is by far the most popular software for this purpose, and platforms such as Github and GitLab have built extensive functionality around projects managed with Git[1].

Figure 4.2 shows the VCS used by respondent NSOs.

Surprisingly, the most common response by far was that the respondent did not use version control software at all for their CPI Production Systems. In fact, almost two thirds of the sample claims to use no VCS whatsoever or file-naming conventions[2].

Our hypothesis here is that (1) some commonly used file formats (e.g., Excel Workbooks) don't lend themselves easily to well-established version control tools like Git and (2) a lack of familiarity with VCS tools in general.

Another interesting observation is that slightly less than one third of the sample uses Git and/or GitHub/GitLab[3].

A very small fraction of the sample indicated using other version control software such as Mercurial, Subversion, or built-in versioning capabilities of another tool/platform.

[1] Note that there is a small sample size caveat with observations involving Other Mix teams. Had the sample size been larger, the observations we observed with this team type may not have been as extreme as what we observed in the survey.

[2] We suspect that this is slightly biased by the fact that the majority of Stream-Aligned teams in our sample are comprised of domain-analysts only rather than having both domain-embedded analysts **and** domain-embedded IT professionals.

[3] For readers who are unfamiliar with the Consumer Prices business domain, these are techniques that are used to calculate period-over-period price changes from a given data source.

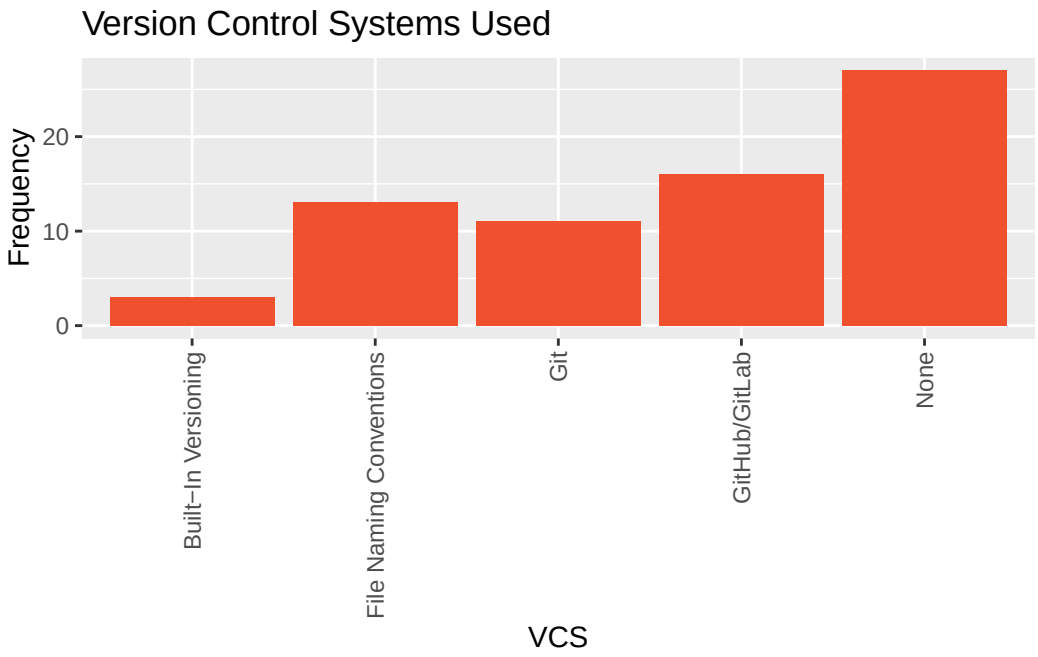


Figure 4.1: VCS used by survey respondents

Having both worked on non-trivial Complex Analytical Systems ourselves, we suspect that adopting industry standard VCS toolingap may greatly reduce the cognitive load for teams who are not currently using any VCS or are only using file-naming conventions to manage source code and documentation for Complex Analytical Systems[4].

4.2 Commercial Software

The next question we asked was which commercial software (if any) is used in each respondent's CPI Production systems.

We were not overly surprised to learn that over two thirds of the respondents listed that Microsoft Excel was used in their CPI Production Systems.

Microsoft Excel satisfies a number of use cases for beginner-friendly tabular data analysis, however, it is not an ideal tool for expressing business logic in Complex Analytical Systems[5].

[4]We are skeptical that these two facts can be true at the same time. Our suspicion is that some respondents have not explicitly thought about version control as a distinct problem with purpose-built tooling that exists to solve it.

[5]In general, the Consumer Price Index is a non-revisable product, meaning that it is not straightforward to "roll back" a change in the same way that would be possible in other settings. Because of this, there is a certain level of due-diligence required for large system changes, meaning there are some limits on how

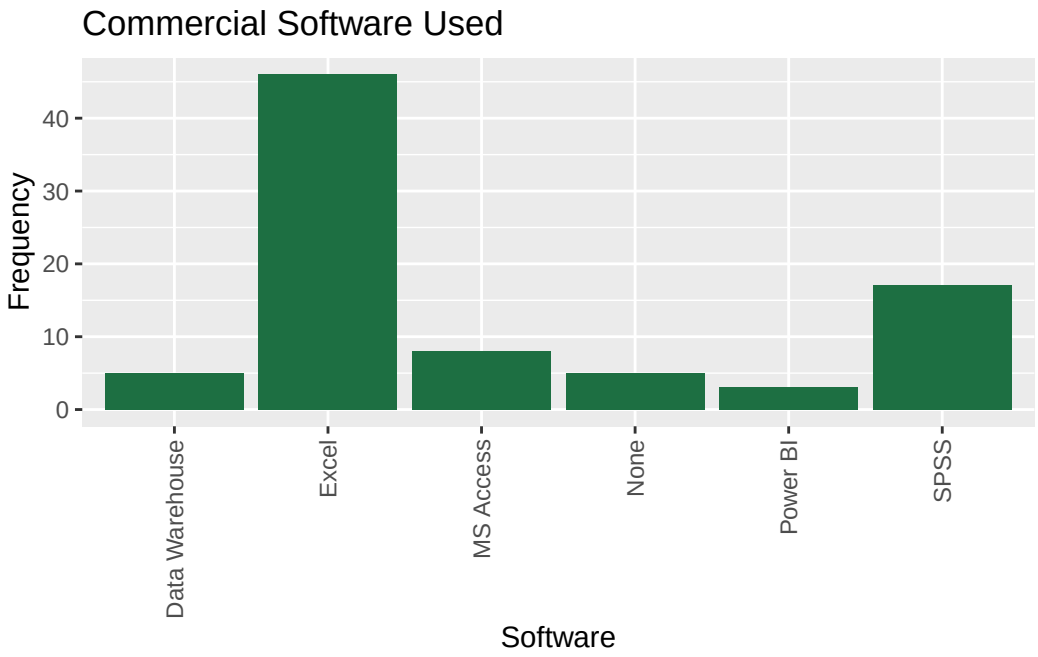


Figure 4.2: VCS used by survey respondents

Some of the key reasons for this include:

- Excel Workbooks are stored in a binary format rather than a plain text format, which doesn't integrate well with Version Control Systems.
- There isn't a well-defined entrypoint to an Excel Workbook (i.e., you can't "run" an Excel workbook the same way you can "run" `python main.py`).
- The business logic encoded into an Excel Workbook is often difficult to read and interpret for any non-trivial Workbook, making Excel Workbooks difficult to maintain as a unit of software.
- Excel Workbooks encourage a high degree of coupling between business logic and data[6].

In a distant second to Microsoft Excel, we find SAS is the next most commonly used commercial product in CPI Production Systems.

frequently CPI Production Systems can be updated.

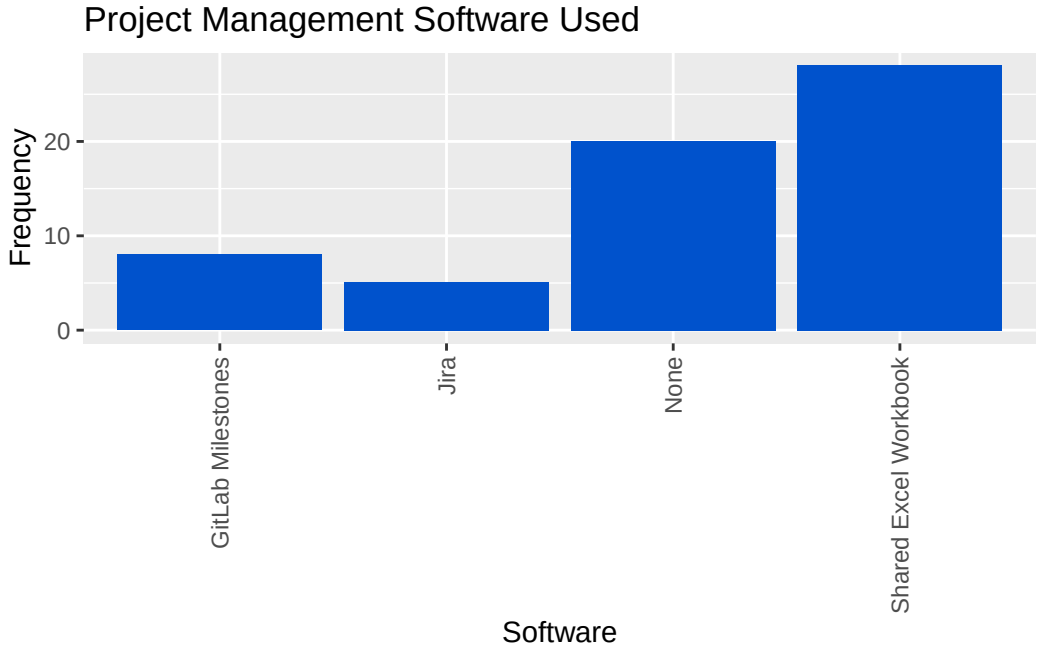
[6]Note that this can be as simple as having a `production` folder and a `development` folder on a network file system and scoping activities to the appropriate folder. More complex separations of development, testing, and production environments are possible, but we encourage readers to start simple if this is a new concept.

4.3 Project Management Software

We asked respondents which project management software they use to manage work done on their CPI Production Systems.

The exact set of features provided by project management software differ depending on the specific software used, but in general, this kind of software is used to plan and coordinate tasks, manage timelines, and track progress on work items.

We believe that using some purpose-built project management software for any non-trivial project is generally a good idea because it encourages a structure to the way projects are managed and it offers a way to reduce cognitive burden for project team members[7].



The most common project management software reported was a shared Excel Workbook, with a bit less than half of respondents stating that they used this as their team's primary means of project management.

Shared Excel Workbooks may be sufficient for keeping track of small tasks, but for any endeavour that requires managing code, data, and documentation changes across multiple individuals, this approach will lack a number of key features to facilitate the project management process.

[7]For example, if there is evidence that certain team structures, team interaction modes, and system architectures are more effective than others, then organizations that adopt the improved team topologies and system architectures may see reduced maintenance costs and faster delivery times, among other benefits.

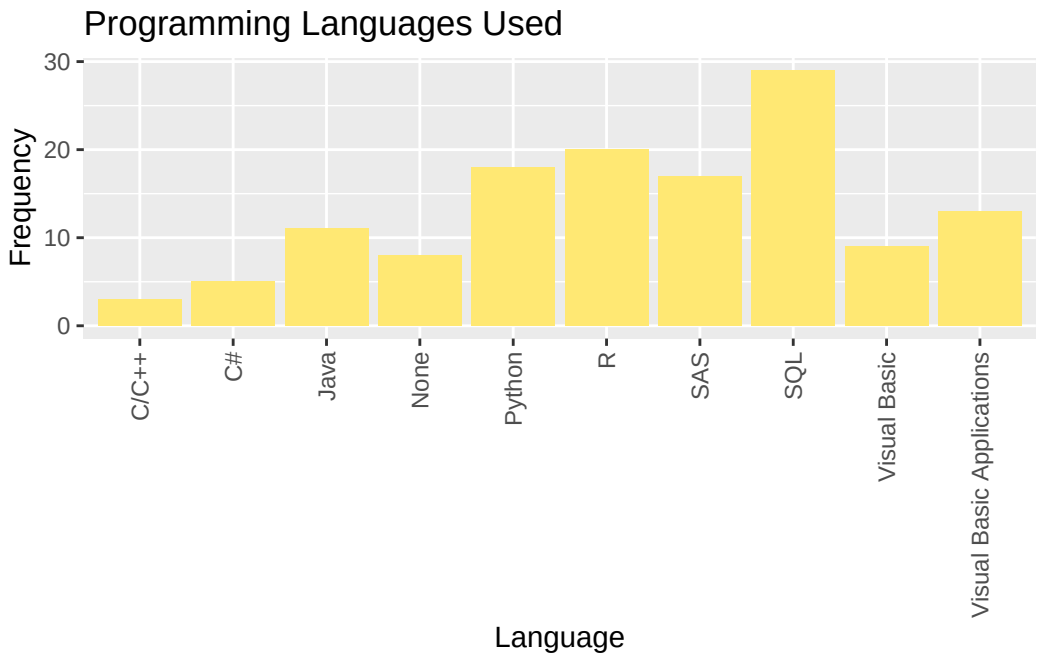
The second most cited answer was not using any project management software whatsoever, with almost one third of respondents indicating this answer.

Given the complexity of CPI Production Systems, we were surprised to see so many individuals not using any project management software. For all but the simplest of initiatives, we believe there is significant value in adopting at least the basic features of some purpose-built project management software[8].

Finally, just over twenty percent of the sample reported using either GitHub Projects, GitLab Milestones, or Jira as their primary software for project management.

4.4 Programming Languages

We asked respondents which programming languages are used to develop their CPI Production Systems. We found several observations noteworthy.



First, just under half of the respondents indicated that they use SQL in the development of their CPI Production Systems.

[8]When working with this quantity of data, it is important to pay attention to the data types of each column and to choose the most parsimonious data types. For example, if 3 columns can be correctly represented with boolean, 16-bit integer, and 32-bit integer data types, there are significant memory savings to be gained by casting the variables to these types upfront.

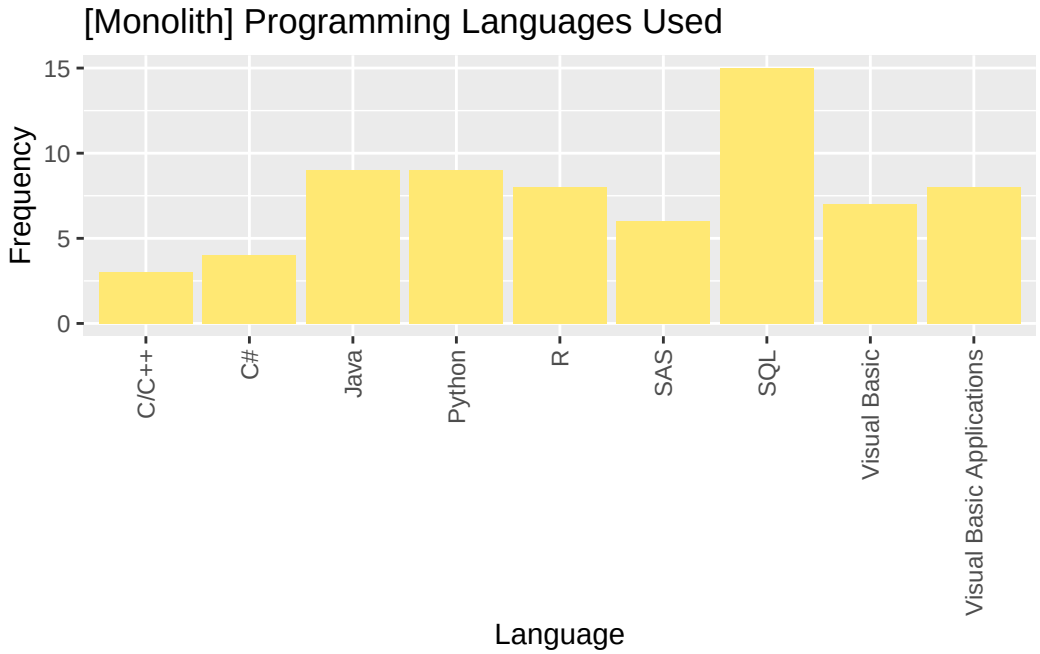
We suspect this answer reflects the fact that SQL still remains a very popular choice of language for expressing tabular data manipulations, as well as the fact that many organizations reported using some kind of database management system (DBMS) in their CPI Production Systems. Our suspicion is that SQL is most commonly used in the ingestion and processing steps in the flow of change[9].

Second, we notice than more than ninety percent of the sample reported using **at least one of** Python, R, and SAS in their CPI Production Systems. Interestingly, less than one quarter of these users reported using Python and/or R but not SAS. In other words, it is quite common for SAS to be used **in conjunction with** Python and/or R rather than being used **instead of** Python and/or R.

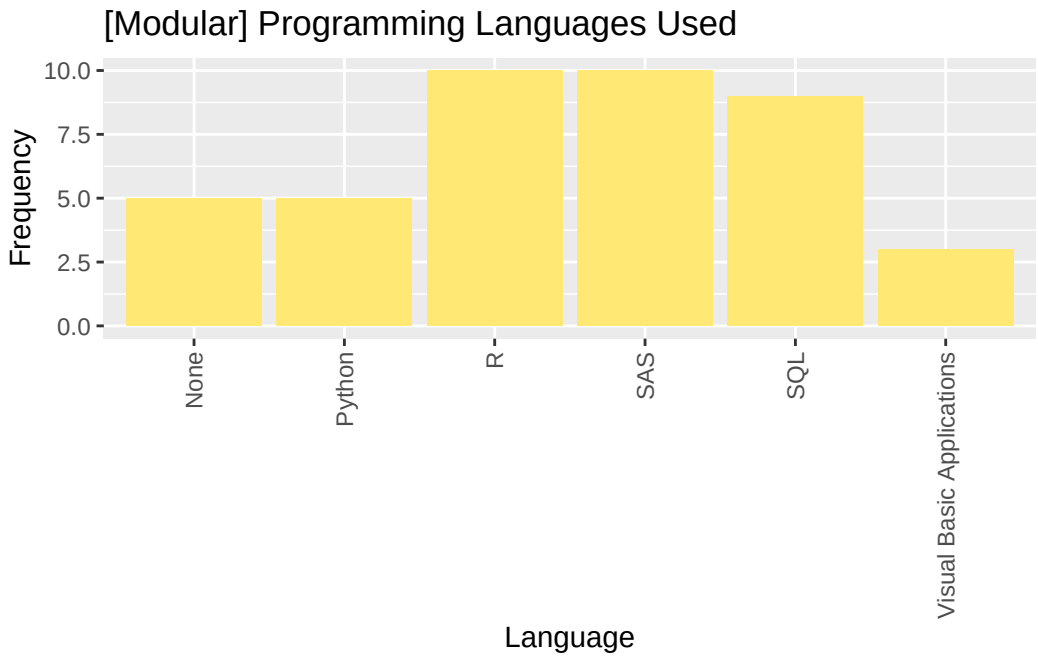
Third, there are some differences in programming languages used between NSOs with monolithic representative systems and NSOs with modular representative systems.

The most noteworthy observation is that Java, C#, C/C++, Visual Basic, and Visual Basic Applications (VBA) are somewhat commonly used by NSOs with monolithic representative systems (almost two thirds) and almost never used by NSOs with modular representative systems (less than one quarter).

Additionally, we notice that slightly over one third of NSOs with monolithic representative systems report using R or SAS, while almost two thirds of NSOs with modular representative systems report using R or SAS.



[9]To minimize response burden, we did not ask respondents to enumerate programming languages used by GSBPM step, so we cannot say for sure in which step(s) SQL is used.



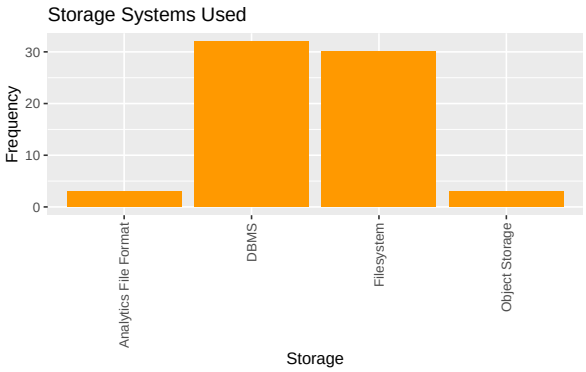
4.5 Data Storage Tools

We conclude the Tools and Technology portion of the survey by asking which data storage tools are in use by respondents in their CPI Production Systems.

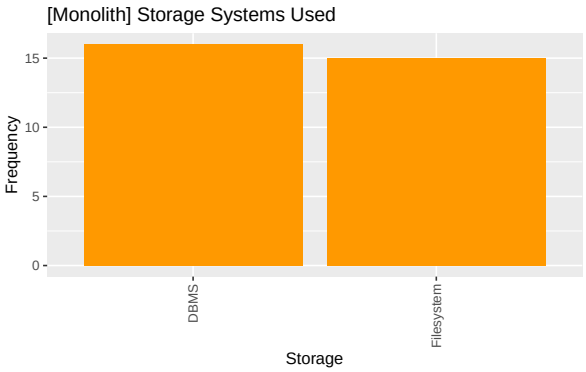
We were surprised by the following findings.

1. A very small percentage of the respondents reported using analytics optimized file formats such as Apache Parquet in their CPI Production Systems. There is a small learning curve associated with using these file formats, but they offer many benefits such as columnar storage on disk, data types supported natively by the file format, and a significantly smaller storage footprint due to columnar data compression strategies[10].
2. There is an approximately equal split between the usage of Database Management Systems (DBMS) and file system storage, which is true across both NSOs with modular representative systems and NSOs with monolithic representative systems.
3. The small number of respondents who reported using something in addition to DBMS or filesystem storage all belonged to NSOs with modular or hybrid representative systems.

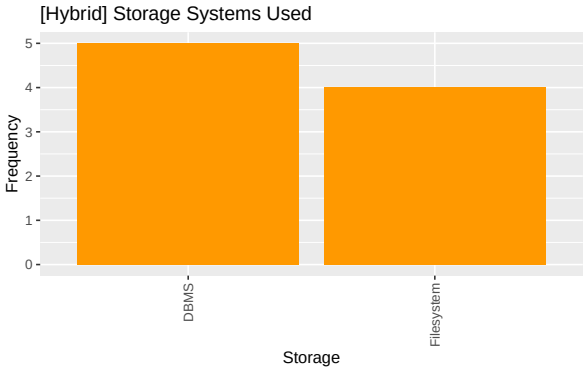
[10] `UPSERT` is a portmanteau of the common `INSERT` and `UPDATE` database operations.



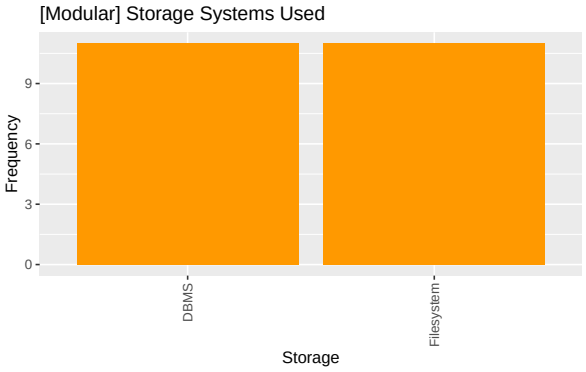
(a) Choice of Storage (Overall)



(a) Choice of Storage (Monolith)



(a) Choice of Storage (Hybrid)



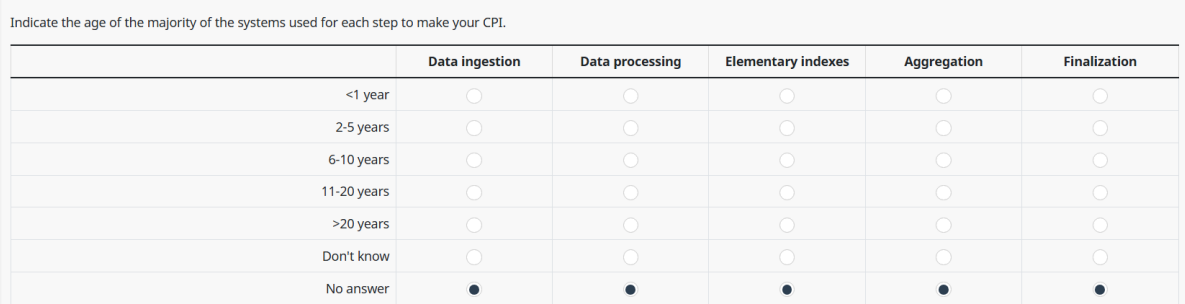
(a) Choice of Storage (Modular)

5 System Age and Updates

The next section of the survey asks questions about (1) the age of the majority of CPI Production Systems and (2) the update frequency of the majority of CPI Production Systems.

5.1 System Age

Respondents are presented with the following question.


Indicate the age of the majority of the systems used for each step to make your CPI.

	Data ingestion	Data processing	Elementary indexes	Aggregation	Finalization
<1 year	○	○	○	○	○
2-5 years	○	○	○	○	○
6-10 years	○	○	○	○	○
11-20 years	○	○	○	○	○
>20 years	○	○	○	○	○
Don't know	○	○	○	○	○
No answer	●	●	●	●	●

Figure 5.1: Age of Systems Survey Question

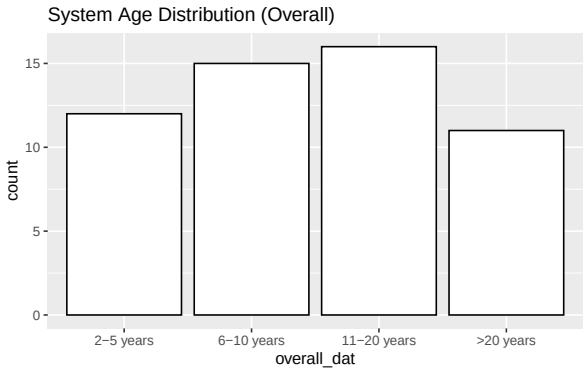
In this question, we aim to understand for how long the **majority** of CPI Production Systems at NSOs have been in operation.

In the context of this question, a system could be long-lived because (1) it was built in an extensible fashion and has been easy to update and maintain over a long time period, (2) the requirements of the system have changed very little over a long time period, or (3) every component of the system has been replaced at some point without ever doing a full "system rewrite" (see Theseus's Paradox).

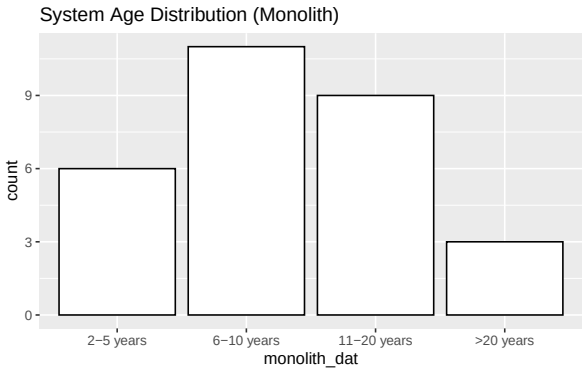
Therefore, without more information about the maintenance history of these systems, we cannot say whether older systems are "good" or "bad". Nevertheless, it is interesting to understand how old the typical CPI Production System is.

We share some notable observations and explanations below.

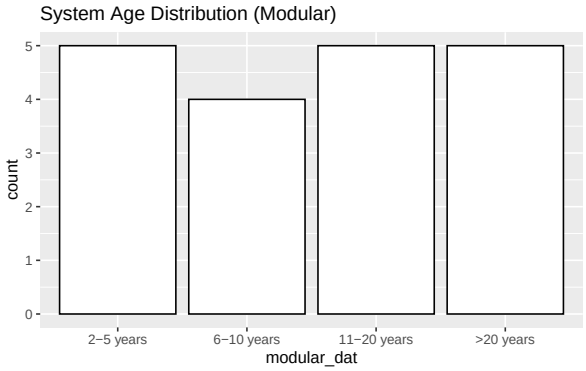
The system age distribution for NSOs with modular representative systems is approximately uniform, while NSOs with monolithic representative systems are more likely to report that the



(a) System Age Distribution (Entire Sample)



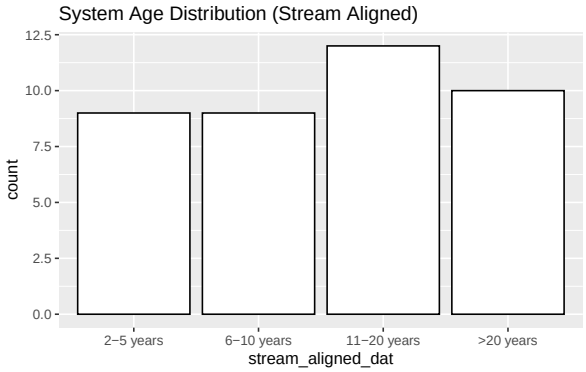
(a) System Age Distribution (Monolith)



(a) System Age Distribution (Modular)

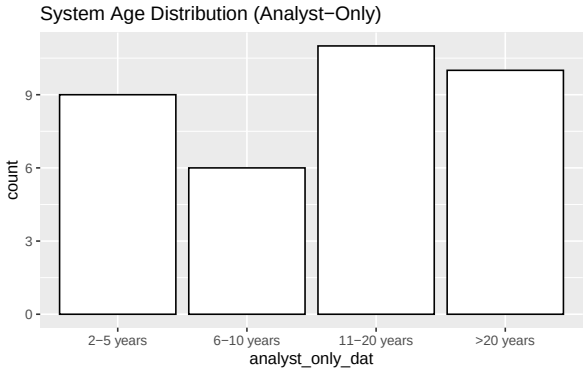
majority of their systems are between 6-20 years old, and less likely to report that the majority of their systems are more than 20 years old.

One possible explanation for this observation is that monolithic systems are more likely to reach a level of complexity where it becomes too difficult to reliably make changes to the system due to a high degree of coupling (inter-dependency) between components[1]. If this is true, it could be the case that relatively few monolithic systems reach an age greater than 20 years before a complete system rewrite is necessary.



(a) System Age Distribution (Stream Aligned)

(a) System Age Distribution (IT-Only)



(a) System Age Distribution (Analyst-Only)

(a) System Age Distribution (Other Mix)

NSOs with IT-Only teams appear relatively less likely to have the majority of systems be less than 20 years old, compared to their Analyst-Only and Stream-Aligned counterparts. One possible explanation is that IT-Only teams are more likely to have the technical skills to perform a full system rewrite when it becomes necessary, although we don't have sufficient data to say for sure.

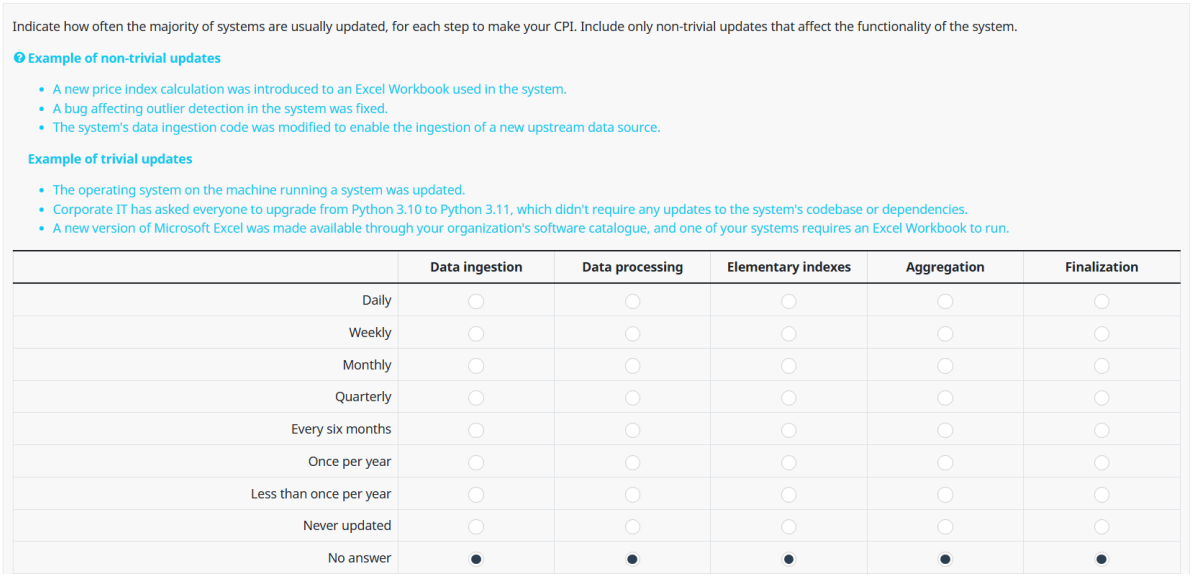
[1]Note that there is a small sample size caveat with observations involving Other Mix teams. Had the sample size been larger, the observations we observed with this team type may not have been as extreme as what we observed in the survey.

Another noteworthy observation is that NSOs with Other Mix teams (see Chapter 3) are most likely to have the majority of systems be between 6-10 years old. It is noteworthy that most of the NSOs with Other Mix teams also have Monolithic representative systems, so there is a high degree of overlap between Other Mix teams and Monolithic representative systems.

Our hypothesis for this observation is that teams comprised of individuals with little shared domain context (e.g., a team with individuals from Corporate IT as well as Domain Analysts) are more likely to produce overly complicated and tightly inter-connected systems that are difficult to change[2]. After 6-10 years, these systems become so difficult to maintain that it becomes necessary to undergo a full system rewrite[3].

5.2 System Update Frequency

Respondents are presented with the following question.


Indicate how often the majority of systems are usually updated, for each step to make your CPI. Include only non-trivial updates that affect the functionality of the system.

Example of non-trivial updates

- A new price index calculation was introduced to an Excel Workbook used in the system.
- A bug affecting outlier detection in the system was fixed.
- The system's data ingestion code was modified to enable the ingestion of a new upstream data source.

Example of trivial updates

- The operating system on the machine running a system was updated.
- Corporate IT has asked everyone to upgrade from Python 3.10 to Python 3.11, which didn't require any updates to the system's codebase or dependencies.
- A new version of Microsoft Excel was made available through your organization's software catalogue, and one of your systems requires an Excel Workbook to run.

	Data ingestion	Data processing	Elementary indexes	Aggregation	Finalization
Daily	○	○	○	○	○
Weekly	○	○	○	○	○
Monthly	○	○	○	○	○
Quarterly	○	○	○	○	○
Every six months	○	○	○	○	○
Once per year	○	○	○	○	○
Less than once per year	○	○	○	○	○
Never updated	○	○	○	○	○
No answer	●	●	●	●	●

Figure 5.9: Age of Systems Survey Question

In this section, we ask respondents how often the majority of CPI Production Systems are usually updated.

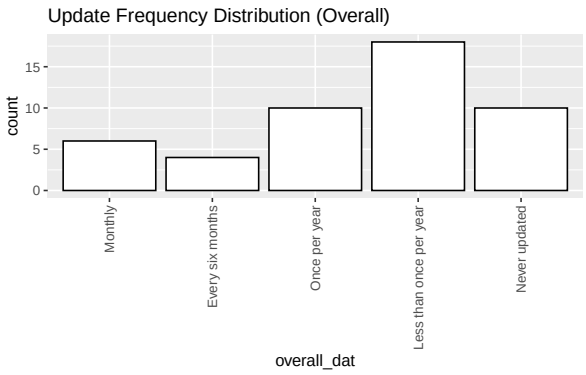
[2]We suspect that this is slightly biased by the fact that the majority of Stream-Aligned teams in our sample are comprised of domain-analysts only rather than having both domain-embedded analysts **and** domain-embedded IT professionals.

[3]For readers who are unfamiliar with the Consumer Prices business domain, these are techniques that are used to calculate period-over-period price changes from a given data source.

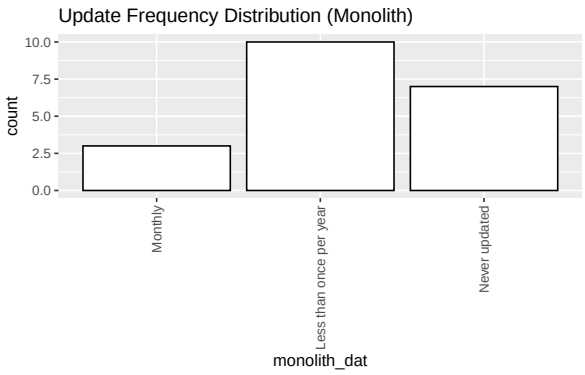
Unless software is being written in a domain where (1) the problem is "solved" (i.e., requirements never change) and/or (2) the software's correctness can be proved mathematically, routinely updating systems (e.g., fixing errors in source code or enhancing the system with new features) is a practical reality of software development.

Given this practical reality, updating systems at a somewhat high frequency is generally considered to be good practice [4].

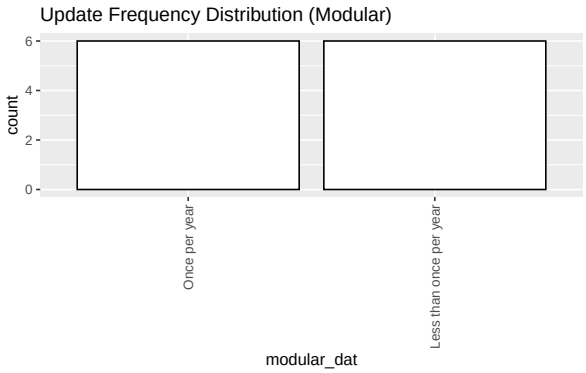
We share below some observations on how frequently respondents update their systems.



(a) Update Frequency Distribution (Overall)



(a) Update Frequency Distribution (Monolith)



(a) Update Frequency Distribution (Modular)

NSOs with monolithic representative systems are much more likely to **never update systems** compared to NSOs with Modular representative systems.

We suspect this is once again related to the fact that monolithic systems tend to be complex and involve many inter-connected components. In the most extreme case, it can be too risky

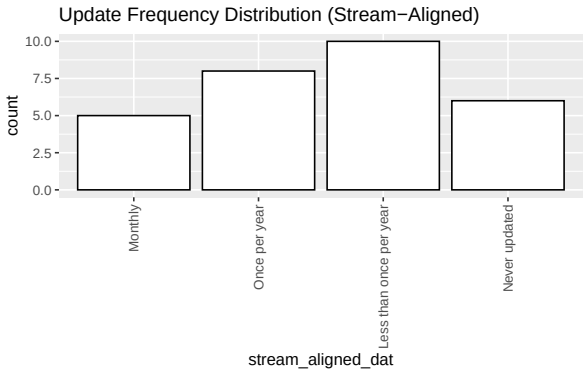
[4] We are skeptical that these two facts can be true at the same time. Our suspicion is that some respondents have not explicitly thought about version control as a distinct problem with purpose-built tooling that exists to solve it.

to update these systems due to the possibility of a change in one part of a system causing unintended consequences in another (seemingly unrelated) part of the system.

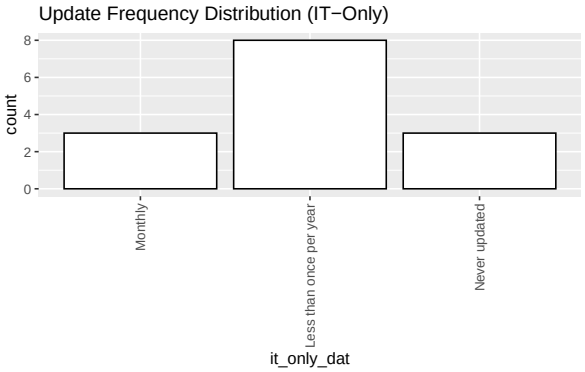
Modular systems, in contrast, tend to be easier to reason about, and involve a lower degree of coupling between system components that are unrelated. Hence, making a change to a system component is less likely to lead to an unintended consequence elsewhere in the system.

Across all respondents, almost one fifth of respondents indicated that the majority of their systems are **never updated**. We are very surprised by this finding.

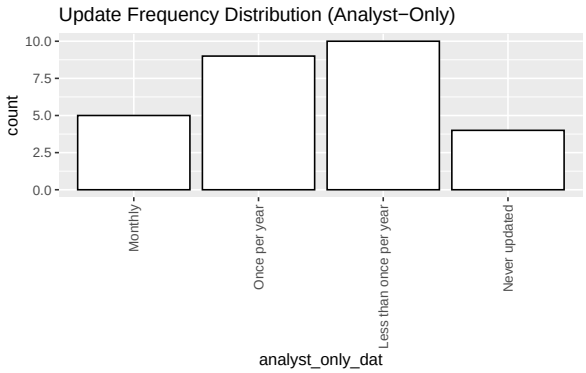
More generally, the distribution of update frequencies seems to be left skewed, with a majority of NSOs having the majority of systems updated once per year or less, or not updated at all.



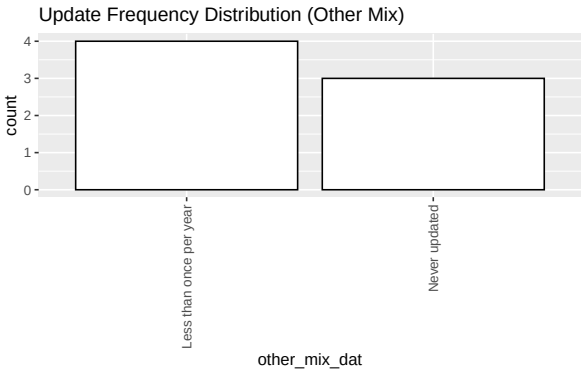
(a) Update Frequency Distribution (Stream-Aligned)



(a) Update Frequency Distribution (IT-Only)



(a) Update Frequency Distribution (Analyst-Only)



(a) Update Frequency Distribution (Other Mix)

When looking at the system age distributions broken down by team type, we do not see much difference from the overall trend. One noteworthy exception to this is the Other Mix team category, which seems to have a relatively high proportion of cases updated less than once per

year or not at all[5].

[5]In general, the Consumer Price Index is a non-revisable product, meaning that it is not straightforward to "roll back" a change in the same way that would be possible in other settings. Because of this, there is a certain level of due-diligence required for large system changes, meaning there are some limits on how frequently CPI Production Systems can be updated.

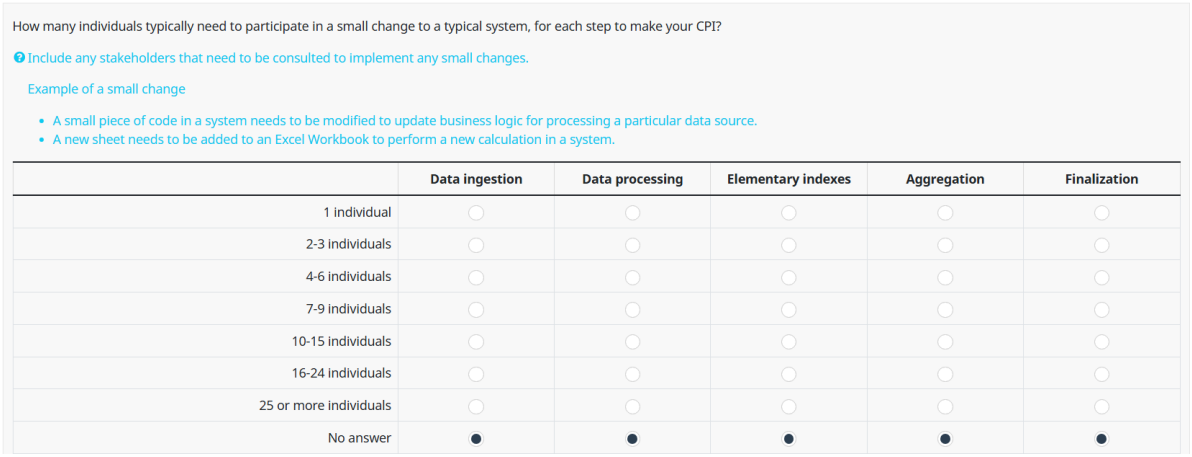
6 Number of People

In this section of the survey, we are interested in how many individuals need to participate in (1) small system changes and (2) large system changes.

The rationale behind asking this question is to get a sense for how much communication overhead is required to make changes to CPI Production Systems. We assume that the requirement of more individuals is associated with more communication overhead.

6.1 Number of People (Small Changes)

Respondents are presented with the following question.



How many individuals typically need to participate in a small change to a typical system, for each step to make your CPI?

Include any stakeholders that need to be consulted to implement any small changes.

Example of a small change

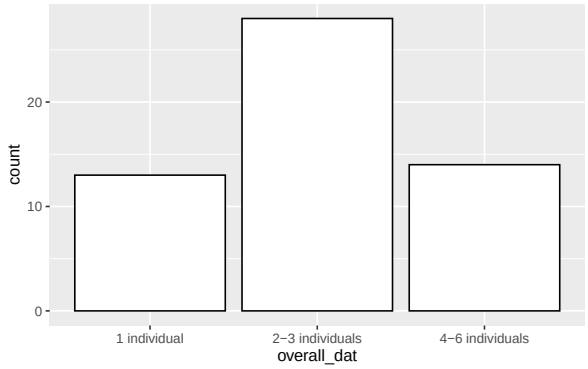
- A small piece of code in a system needs to be modified to update business logic for processing a particular data source.
- A new sheet needs to be added to an Excel Workbook to perform a new calculation in a system.

	Data ingestion	Data processing	Elementary indexes	Aggregation	Finalization
1 individual	○	○	○	○	○
2-3 individuals	○	○	○	○	○
4-6 individuals	○	○	○	○	○
7-9 individuals	○	○	○	○	○
10-15 individuals	○	○	○	○	○
16-24 individuals	○	○	○	○	○
25 or more individuals	○	○	○	○	○
No answer	●	●	●	●	●

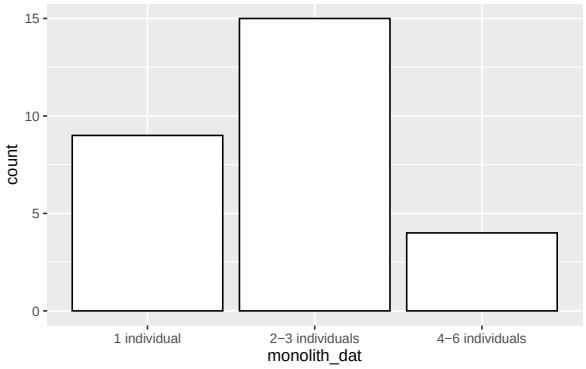
Figure 6.1: Number of Individuals for small changes

Overall, the majority of respondents indicate between 1-3 individuals needing to be involved with small changes, while a minority indicate that 4 or more individuals need to be involved with small changes.

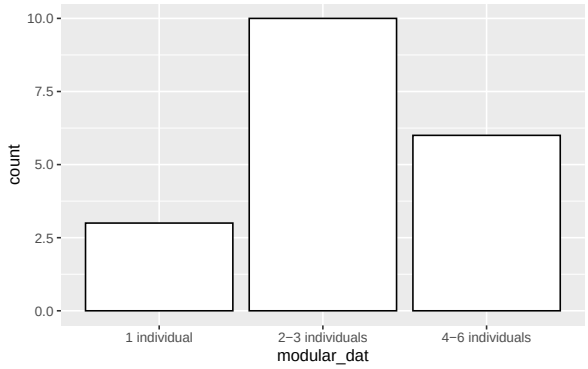
Our sense is that 1-3 individuals participating in a small change is reasonable. For low risk small changes, one person could make the change in isolation, while for more important small changes, one or two individuals could quickly peer review the change before it is implemented.



(a) Number of Individuals for Small Changes (Overall)



(a) Number of Individuals for Small Changes (Monolith)



(a) Number of Individuals for Small Changes (Modular)

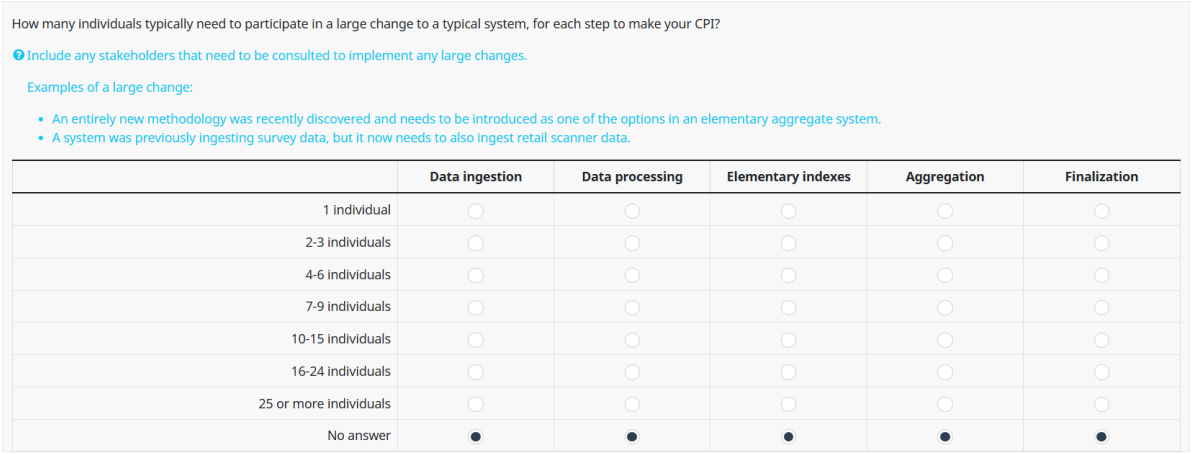
It appears that NSOs with Hybrid or Modular representative systems are slightly more likely to involve 4-6 individuals in small changes compared to NSOs with Monolithic representative systems.

Our suspicion here is that NSOs with Hybrid or Modular representative systems are more likely to have "interface boundaries" between systems maintained by two or more distinct teams. If this is the case, certain small changes may require input from individuals across two or more teams. This is not necessarily unreasonable. If system and team boundaries are well defined, communication about small changes to a system can still be efficient even if a slightly greater number of individuals need to be made aware of the small change.

There are no significant differences in the number of individuals required for small changes between the various team types.

6.2 Number of People (Large Changes)

Respondents are presented with the following question.


How many individuals typically need to participate in a large change to a typical system, for each step to make your CPI?

Include any stakeholders that need to be consulted to implement any large changes.

Examples of a large change:

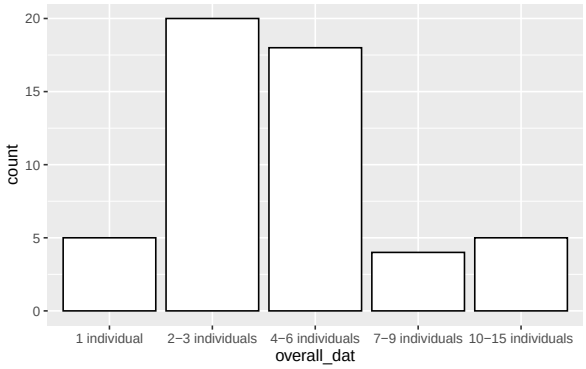
- An entirely new methodology was recently discovered and needs to be introduced as one of the options in an elementary aggregate system.
- A system was previously ingesting survey data, but it now needs to also ingest retail scanner data.

	Data ingestion	Data processing	Elementary indexes	Aggregation	Finalization
1 individual	○	○	○	○	○
2-3 individuals	○	○	○	○	○
4-6 individuals	○	○	○	○	○
7-9 individuals	○	○	○	○	○
10-15 individuals	○	○	○	○	○
16-24 individuals	○	○	○	○	○
25 or more individuals	○	○	○	○	○
No answer	●	●	●	●	●

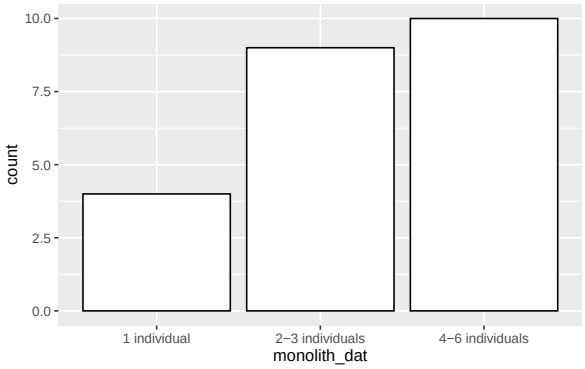
Figure 6.5: Number of Individuals for large changes

Unsurprisingly, more individuals are required to participate in large changes than small changes. It appears that the majority of respondents require between 2-6 individuals to participate in a large change to a system.

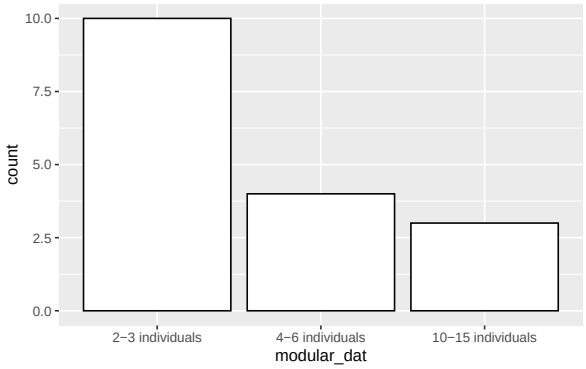
Interestingly, there is not a significant difference in the number of people required for large changes between NSOs with Monolithic representative systems and NSOs with Modular representative systems. Moreover, there is also no meaningful difference in this metric between the various team compositions.



(a) Number of Individuals for Large Changes (Overall)



(a) Number of Individuals for Large Changes (Monolith)



(a) Number of Individuals for Large Changes (Modular)

We were a bit surprised by this finding, as we expected that certain team compositions and system architectures would be associated with differing numbers of people who need to participate in large changes.

Our hypothesis here is that the number of people required to participate in large changes is probably a function of organization size more than anything else. For example, if the size of all teams involved in a large change for a small NSO is 6, then it would be impossible for the total number of individuals to exceed 6, regardless of the CPI System architecture or the team compositions.

7 Lead Time

In this section, we ask questions about the lead time of (1) small and (2) large changes to CPI Production Systems.

In the context of software systems, lead time measures the time it takes for a committed code change to reach production, which reflects the efficiency of the software delivery process (Forsgren, Humble, and Kim (2018)).

Given that we are looking at Complex Analytical Systems rather than traditional software systems in this survey (see Section 1.2), we modify this definition slightly to refer to the total amount of time required to get an end-to-end change to a CPI Production System implemented, which could include activities in addition to code implementation such as data analysis, methodology research, and discussions with data providers.

Lead time is generally regarded as a very important performance metric, and short lead times are generally considered better than long lead times[1].

We prompt the respondent with some examples of small and large changes before asking the lead time questions.

Example of a small change:

- A small piece of code in a system needs to be modified to update business logic for processing a particular data source.
- A new sheet needs to be added to an Excel Workbook to perform a new calculation in a system.

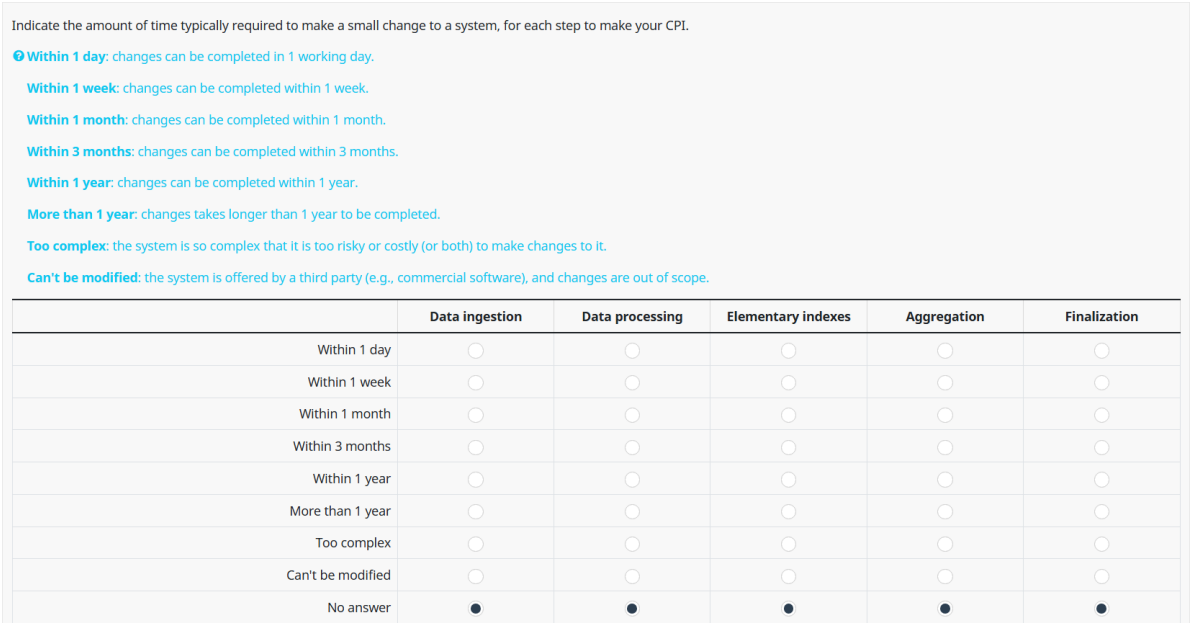
Examples of a large change:

- An entirely new methodology was recently discovered and needs to be introduced as one of the options in an elementary aggregate system.
- A system was previously ingesting survey data, but it now needs to also ingest retail scanner data.

[1]Note that there is a small sample size caveat with observations involving Other Mix teams. Had the sample size been larger, the observations we observed with this team type may not have been as extreme as what we observed in the survey.

7.1 Lead Time (Small Changes)

Respondents are presented with the following question.


Indicate the amount of time typically required to make a small change to a system, for each step to make your CPI.

Within 1 day: changes can be completed in 1 working day.

Within 1 week: changes can be completed within 1 week.

Within 1 month: changes can be completed within 1 month.

Within 3 months: changes can be completed within 3 months.

Within 1 year: changes can be completed within 1 year.

More than 1 year: changes takes longer than 1 year to be completed.

Too complex: the system is so complex that it is too risky or costly (or both) to make changes to it.

Can't be modified: the system is offered by a third party (e.g., commercial software), and changes are out of scope.

	Data ingestion	Data processing	Elementary indexes	Aggregation	Finalization
Within 1 day	○	○	○	○	○
Within 1 week	○	○	○	○	○
Within 1 month	○	○	○	○	○
Within 3 months	○	○	○	○	○
Within 1 year	○	○	○	○	○
More than 1 year	○	○	○	○	○
Too complex	○	○	○	○	○
Can't be modified	○	○	○	○	○
No answer	●	●	●	●	●

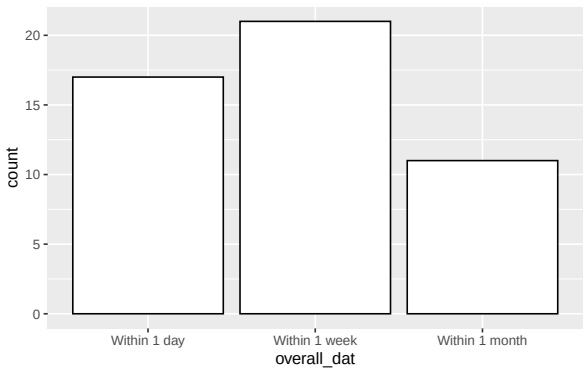
Figure 7.1: Lead times for small changes

Overall, the majority of lead times for small changes are between 1 day and 1 week, with a minority of respondents indicating small changes happening within 3 months (i.e., 3 months or less).

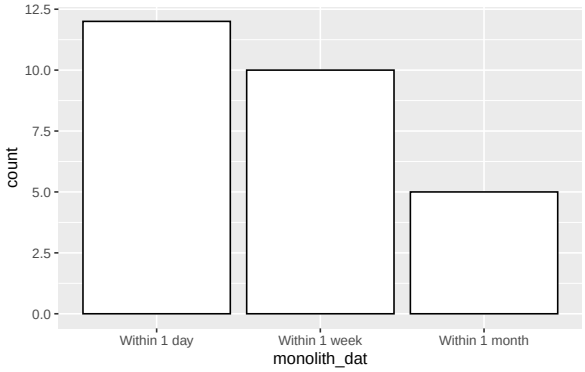
Our expectation is that small changes of the magnitude we described in the question prompt should take teams at most a few days to implement, test, and integrate into production systems. We were a bit surprised to see a non-trivial fraction of the sample reporting lead times of "Within 1 month" or "Within 3 months" for small changes.

It is also noteworthy that NSOs with monolithic representative systems were more likely to report lead times for small changes of "Within 1 day", whereas NSOs with modular representative systems were more likely to report lead times for small changes of "Within 1 week".

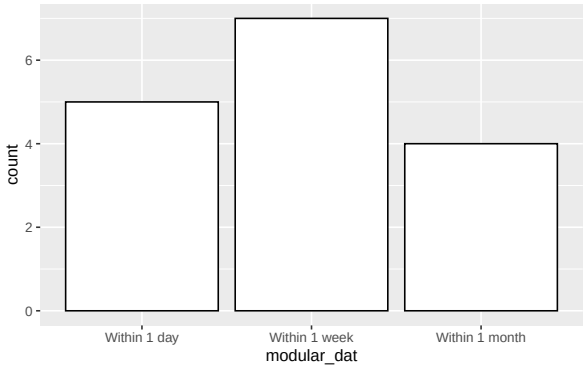
Our hypothesis here is that, by definition, a modular representative system is more likely than a monolithic representative system to span two or more teams. Therefore, small changes may still need to be reviewed by a member on each team. It is reasonable that it might take more than one business day to find a time where the various team members are available to meet, so this alone could cause a small change to take more than one day.



(a) Lead Time for Small Changes (Overall)

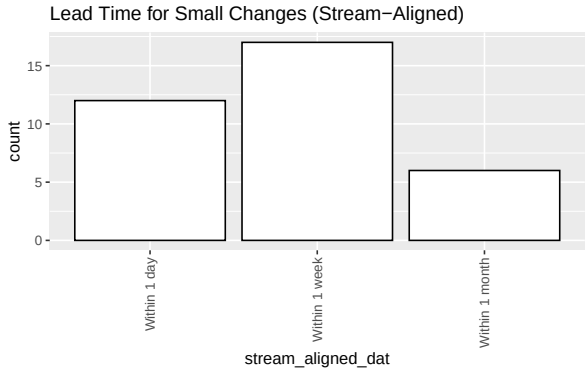


(a) Lead Time for Small Changes (Monolith)

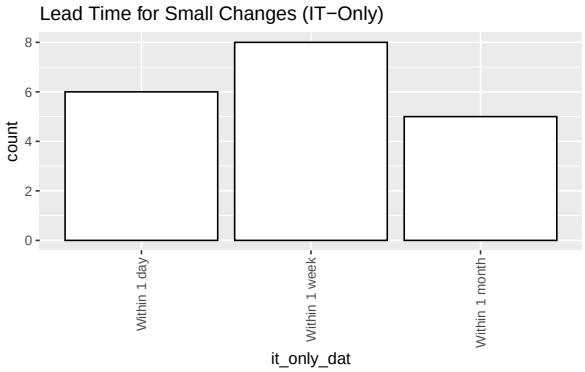


(a) Lead Time for Small Changes (Modular)

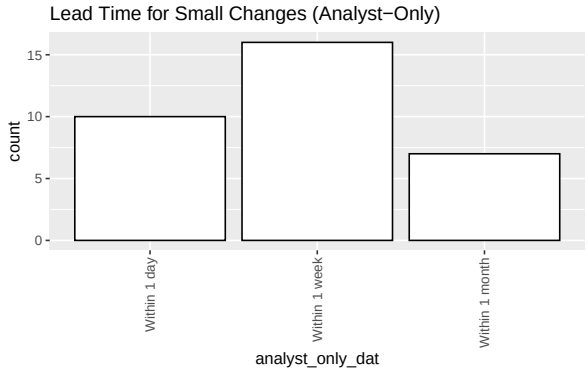
In either case, our view is that "Within 1 day" and "Within 1 week" are both reasonable answers to this question.



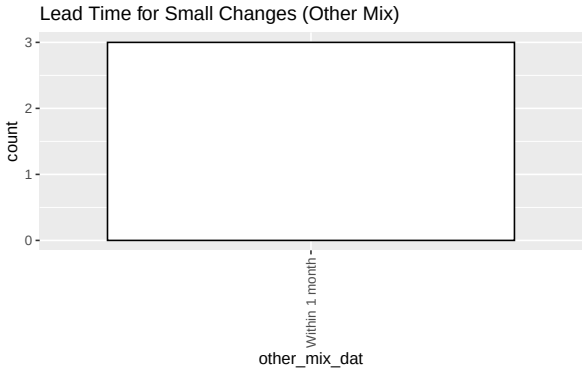
(a) Lead Time for Small Changes (Stream-Aligned)



(a) Lead Time for Small Changes (IT-Only)



(a) Lead Time for Small Changes (Analyst-Only)



(a) Lead Time for Small Changes (Other Mix)

With respect to team composition, Stream-Aligned teams, IT-Only teams, and Analyst-Only teams had lead time distributions that were close to the overall lead time distribution. The most common response for Other Mix teams is "Within 1 month", but the sample size for this team type is too small to conclude anything meaningful here.

7.2 Lead Time (Large Changes)

Respondents are presented with the following question.

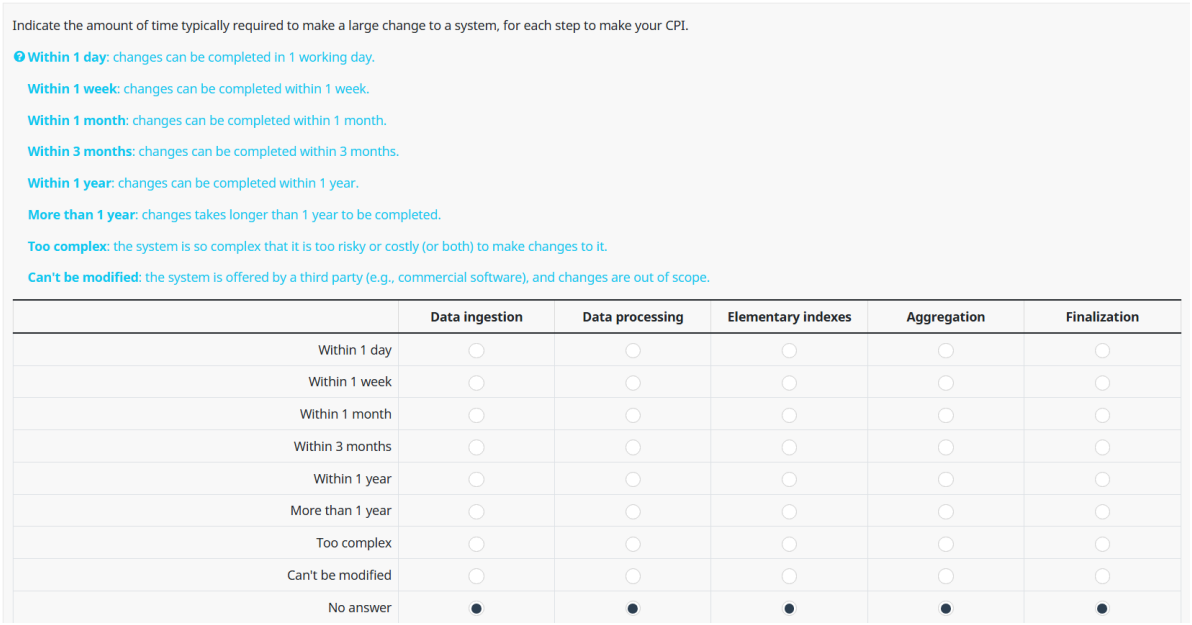


Figure 7.9: Lead times for large changes

Overall, the lead time for large changes was very right skewed, with the most common response being that large changes happen "Within 1 month".

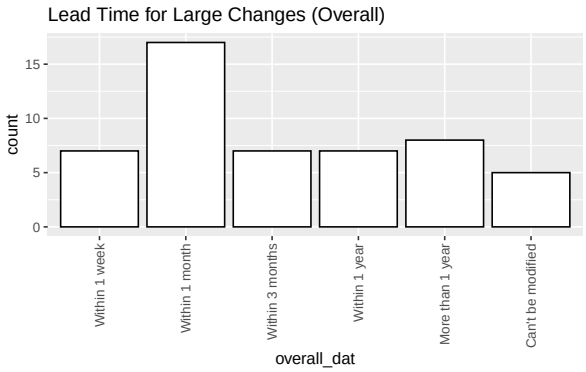
It is interesting that, unlike with small changes, NSOs with Monolithic representative systems are more likely to report lead times of "Within 1 year" or "More than 1 year", and they are also more likely to report an answer of "Too complex" or "Can't be modified"[2].

This result is consistent with industry knowledge (Ford et al. (2021), Richards and Ford (2020)) as well as earlier findings in this report (e.g., sec-age). Monolithic systems are more likely to have components that are highly coupled (inter-connected) compared to modular systems. This high degree of coupling not only makes it more difficult to reason through changes to the system, but it also requires more rigorous testing to ensure that a large change doesn't break a seemingly unrelated component. Therefore, it is not surprising to observe that NSOs with monolithic representative systems are more likely to report higher lead times compared to NSOs with Modular representative systems.

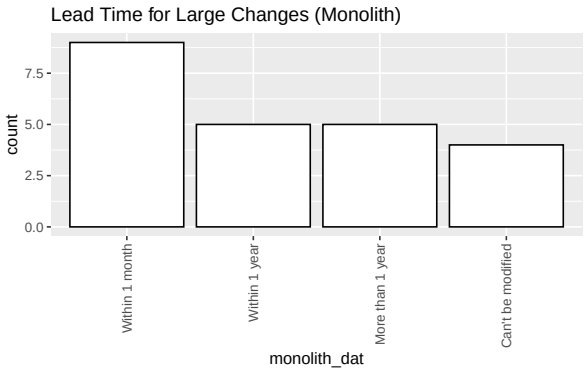
When looking at the distribution of lead times for large changes by team type, there are several noteworthy observations.

First, Stream-Aligned teams and Analyst-Only teams were more likely to report lead times of "Within 1 month" compared to any other answer. Stream-Aligned teams and Analyst-Only

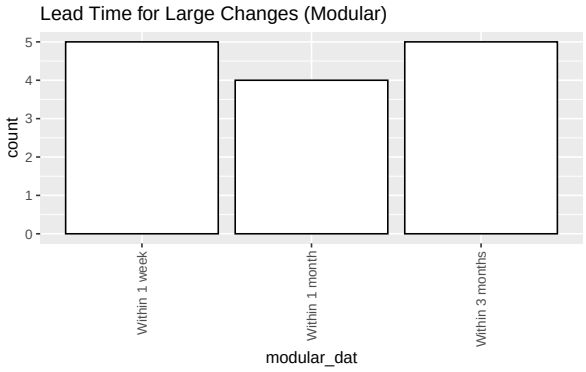
[2]For readers who are unfamiliar with the Consumer Prices business domain, these are techniques that are used to calculate period-over-period price changes from a given data source.



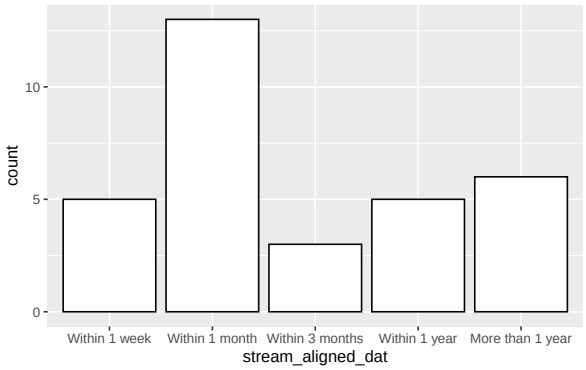
(a) Lead Time for Large Changes (Overall)



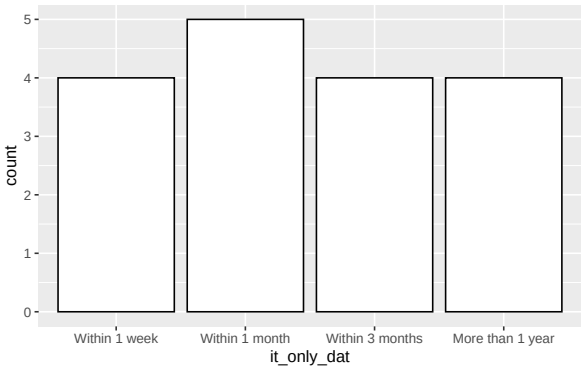
(a) Lead Time for Large Changes (Monolith)



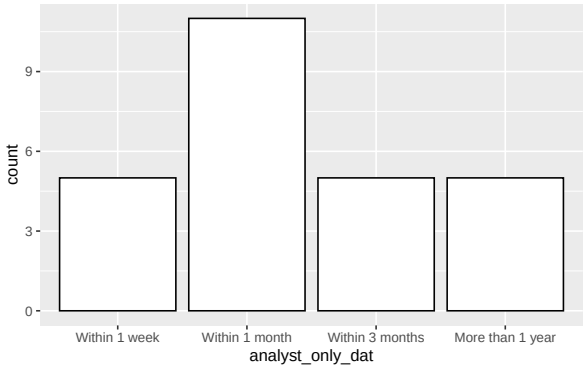
(a) Lead Time for Large Changes (Modular)



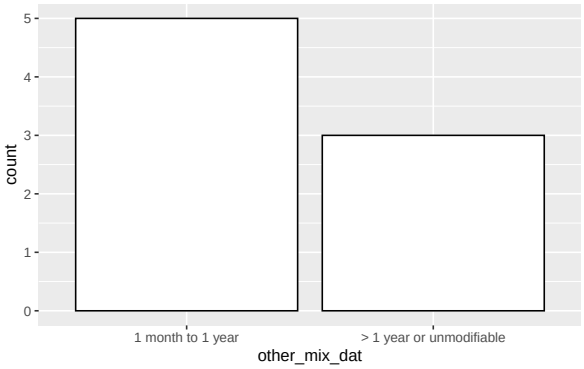
(a) Lead Time for Small Changes (Stream-Aligned)



(a) Lead Time for Small Changes (IT-Only)



(a) Lead Time for Small Changes (Analyst-Only)



(a) Lead Time for Small Changes (Other Mix)

teams were a bit more likely to report shorter lead times compared to IT-Only teams, and were much more likely to report shorter lead times compared to Other Mix teams.

This finding is consistent with the work of Skelton, Pais, and Malan (2019), which suggests that Stream-Aligned teams organized around an end-to-end slice of a particular business domain tend to move faster[3].

It is also worth noting that "More than 1 year" is a somewhat common answer across **all** team types, suggesting that a fraction of NSOs probably struggle with high lead times for reasons other than team types.

We hypothesize that the lack of shared domain context in Other Mix teams could explain why their reported lead times are much higher than the other 3 team types. As discussed in Chapter 5, the lack of a shared domain context increases communication overhead as all parties involved need to spend additional time bridging knowledge gaps. This extra communication overhead could extend each step of the system development process, leading to longer lead times overall.

[3]We suspect that this is slightly biased by the fact that the majority of Stream-Aligned teams in our sample are comprised of domain-analysts only rather than having both domain-embedded analysts **and** domain-embedded IT professionals.

8 Alternative Data

In the world of official statistics, the term **alternative data** is used to describe sources of data other than traditional field-collected data that can be used to derive meaningful statistics.

These data sources often require more complicated tooling and methodology to work with, but offer certain benefits over field-collected data such as collection cost and comprehensiveness[1].

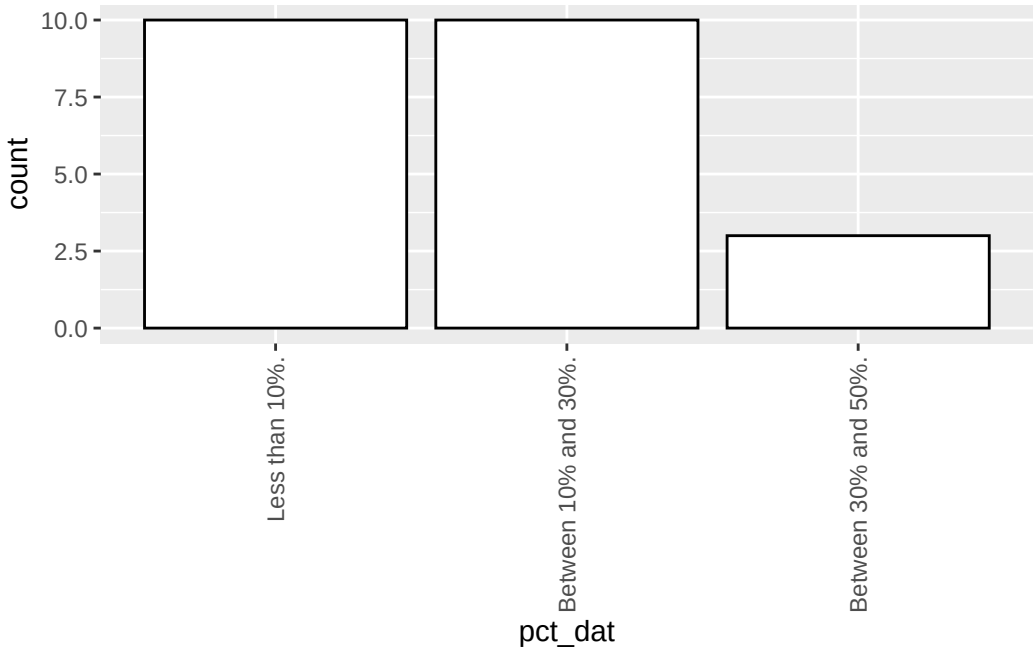
In this section, we ask respondents about the extent to which they currently leverage alternative data sources in their CPI Production Systems. We also ask respondents about the primary challenges they face with respect to the adoption of Alternative Data in their CPI Production Systems.

8.1 Alternative Data Usage

Just under two thirds of respondents report not using alternative data sources at all. Of those that use alternative data sources, the majority of respondents report that "Less than 10%" or "Between 10% and 30%" of their CPI is derived from alternative data sources[2].

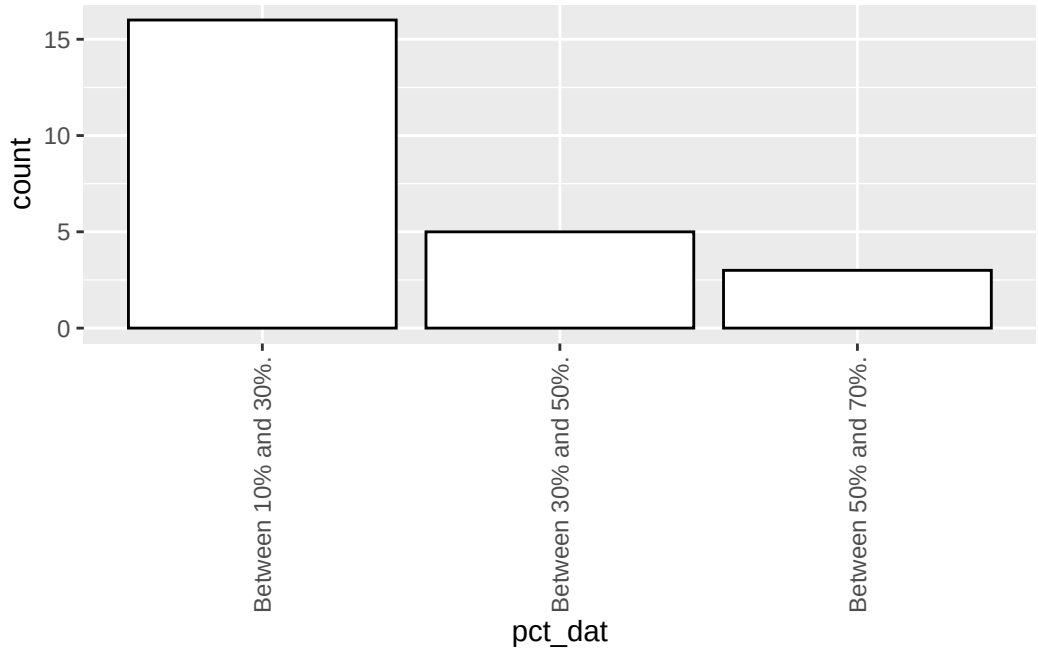
[1] Note that there is a small sample size caveat with observations involving Other Mix teams. Had the sample size been larger, the observations we observed with this team type may not have been as extreme as what we observed in the survey.

[2] We suspect that this is slightly biased by the fact that the majority of Stream-Aligned teams in our sample are comprised of domain-analysts only rather than having both domain-embedded analysts **and** domain-embedded IT professionals.



Of those NSOs that don't currently use alternative data sources, almost three quarters of them report that they would like to use alternative data sources. Of those NSOs that would like to use alternative data sources, we asked them how much alternative data they would like to use in their CPI Production Systems in an ideal scenario. We show this distribution below.

```
Warning in geom_bar(binwidth = 1, colour = "black", fill = "white"): Ignoring
unknown parameters: `binwidth`
```



It appears that most NSOs want between 10% and 50% of their CPI to be comprised of alternative data sources (by expenditure weight). Our suspicion is that leveraging alternative data may be challenging for certain components of the CPI, so even in an ideal scenario, NSOs may prefer to continue using field collected data for quality reasons.

8.2 Which Price Index Methods are Used on Alternative Data Sources

Of those respondents reporting that they currently use alternative data sources, we asked which price index methods are most commonly used[3]. The table below summarizes the number of respondents who are using each method[4].

GEKS	6
Time Dummy Hedonic	4
Hedonic	6
Other Multilateral	< 3
Dynamic Sample	5

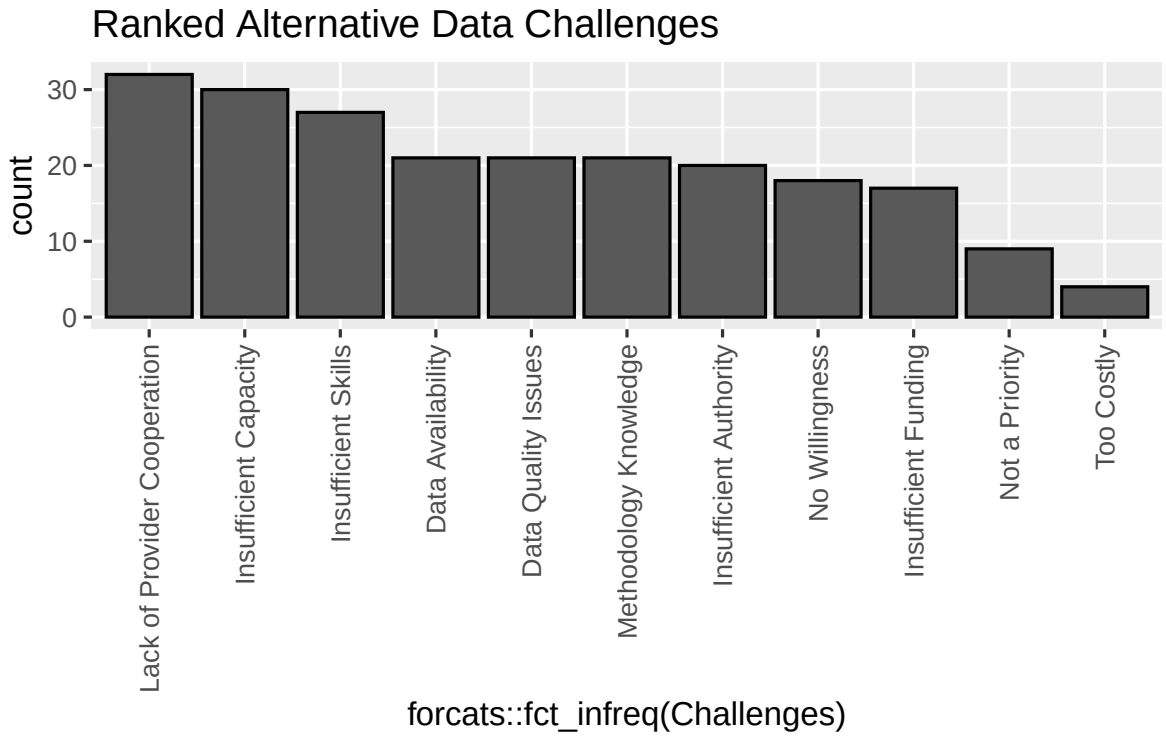
[3]For readers who are unfamiliar with the Consumer Prices business domain, these are techniques that are used to calculate period-over-period price changes from a given data source.

[4]We are skeptical that these two facts can be true at the same time. Our suspicion is that some respondents have not explicitly thought about version control as a distinct problem with purpose-built tooling that exists to solve it.

Fixed Sample	16
Other	7

8.3 What Challenges are Faced in the Adoption of Alternative Data

We asked all respondents to rank the challenges they face in adopting alternative data sources(regardless of whether or not they are currently using alternative data in their CPI Production Systems).



(a) Ranked Alternative Data Challenges

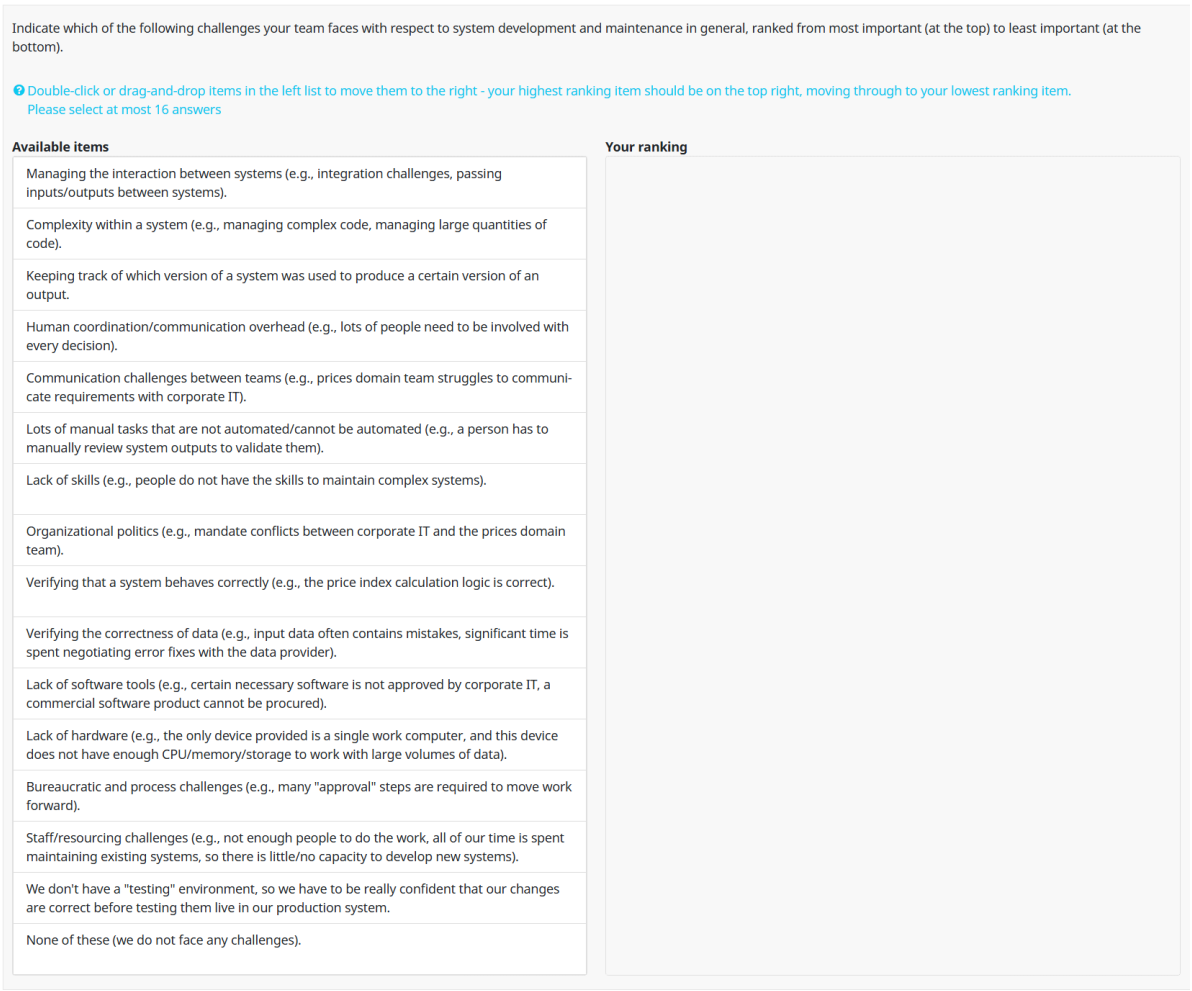
Some alternative data challenges will be region-specific, such as lack of cooperation from data providers, authority to collect alternative data, and availability of alternative data.

However, some commonly cited issues such as Insufficient Skills and Methodology Knowledge can be addressed in part by effective knowledge sharing from domain experts.

9 Challenges

We conclude the survey by asking respondents to rank the challenges they face with respect to system development and maintenance in general.

The survey question presented to respondents was as follows.



Indicate which of the following challenges your team faces with respect to system development and maintenance in general, ranked from most important (at the top) to least important (at the bottom).

Double-click or drag-and-drop items in the left list to move them to the right - your highest ranking item should be on the top right, moving through to your lowest ranking item.
Please select at most 16 answers

Available items

- Managing the interaction between systems (e.g., integration challenges, passing inputs/outputs between systems).
- Complexity within a system (e.g., managing complex code, managing large quantities of code).
- Keeping track of which version of a system was used to produce a certain version of an output.
- Human coordination/communication overhead (e.g., lots of people need to be involved with every decision).
- Communication challenges between teams (e.g., prices domain team struggles to communicate requirements with corporate IT).
- Lots of manual tasks that are not automated/cannot be automated (e.g., a person has to manually review system outputs to validate them).
- Lack of skills (e.g., people do not have the skills to maintain complex systems).
- Organizational politics (e.g., mandate conflicts between corporate IT and the prices domain team).
- Verifying that a system behaves correctly (e.g., the price index calculation logic is correct).
- Verifying the correctness of data (e.g., input data often contains mistakes, significant time is spent negotiating error fixes with the data provider).
- Lack of software tools (e.g., certain necessary software is not approved by corporate IT, a commercial software product cannot be procured).
- Lack of hardware (e.g., the only device provided is a single work computer, and this device does not have enough CPU/memory/storage to work with large volumes of data).
- Bureaucratic and process challenges (e.g., many "approval" steps are required to move work forward).
- Staff/resourcing challenges (e.g., not enough people to do the work, all of our time is spent maintaining existing systems, so there is little/no capacity to develop new systems).
- We don't have a "testing" environment, so we have to be really confident that our changes are correct before testing them live in our production system.
- None of these (we do not face any challenges).

Your ranking

We show a stacked bar chart of the 5 most commonly cited challenges below.

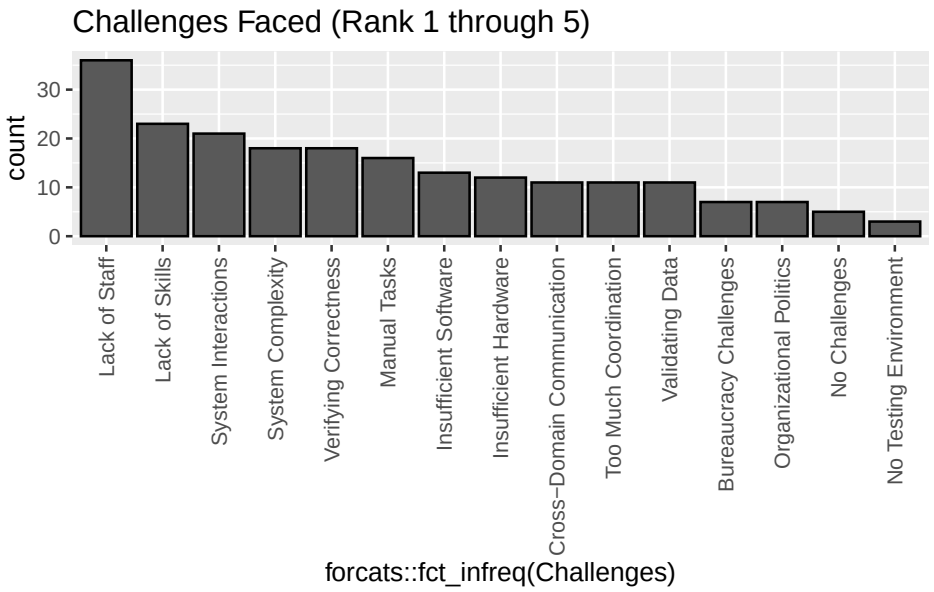


Figure 9.1: Challenges Faced (Rank 1 through 5)

By far the most commonly cited challenge was "Lack of Staff". It is important to note that this reported challenge could be explained by any combination of (1) teams working at peak efficiency who are bottle necked by the number of individuals, or (2) teams where the same number of individuals could improve their efficiency if other challenges were alleviated (e.g., skills gaps, technology gaps, and so on).

The second most commonly cited challenge was "Lack of Skills" in the development and maintenance of CPI Production Systems. Given the technical, bespoke, and multidisciplinary nature of CPI Production Systems, there is probably a significant opportunity to bridge the skills gap by finding ways to facilitate knowledge sharing with domain experts.

Interestingly, managing the interaction between systems ("System Interactions") registered as the third most common challenge. Our suspicion is that most people who develop and maintain CPI Production Systems don't think extensively about how to integrate the outputs from one system into the inputs of another downstream system.

For example, if one analyst writes an R script that performs some data processing and writes an intermediate output to an `intermediate-table.RData` file, and a second analyst later discovers they can leverage that intermediate output in another system they are working on, it is unlikely that the second analyst will simply read the `intermediate-table.RData` file and it will seamlessly integrate with their system.

The more likely scenario is that the first analyst's output table will not be immediately compatible with the second analyst's system (e.g., column names don't match, data types for a

primary key column don't match, the two analysts were using different semantics for the same concept, and so on)[1].

In our experience, these kinds of system integration issues are usually small enough that they are resolved in an ad hoc manner. However, the accumulation of many ad hoc solutions over time can create significant technical debt and lead to significant system interaction challenges.

Tied for fourth place are (1) complexity within a system (e.g., managing large quantities of code) and (2) verifying that a system behaves correctly.

Maintaining large and complex code bases over time is an inherently challenging problem that requires years of practice. Our suspicion is that applying a relatively small number of relevant ideas from software engineering can lead to a significant reduction in the complexity of CPI Production Systems[2]. However, we emphasize that mastering these concepts takes significant practice, even if the concepts are fairly straightforward at face value.

We were not surprised to learn that verifying system correctness registered as one of the top challenges due to the complexity of CPI Production Systems. While there are a large number of tools and techniques that can greatly facilitate the testing of software, being able to take advantage of these tools and techniques requires a minimum level of knowledge. Every major programming language comes with at least one comprehensive framework for automated testing. However, taking advantage of these frameworks require adopting certain idioms in order to integrate the testing framework into the code under test[3]. In brief, very good solutions to this challenge exist, but they require a baseline level of programming skills to adopt in the first place.

As a final note, we were very surprised to see so few respondents identifying Version Control as a challenge. In Chapter 4, the majority of respondents indicated that they either don't use a version control system at all, or they use file-naming conventions to version files. We are skeptical that these two facts are true at the same time.

Our suspicion is that the concept of learning how to use a tool like Git that exists specifically for the purpose of revision control is not something that many survey respondents considered until encountering it in the survey. We therefore suspect that respondents may have lumped revision control challenges with another type of challenge such as "Manual Tasks" or "System Complexity".

[1] Note that there is a small sample size caveat with observations involving Other Mix teams. Had the sample size been larger, the observations we observed with this team type may not have been as extreme as what we observed in the survey.

[2] We suspect that this is slightly biased by the fact that the majority of Stream-Aligned teams in our sample are comprised of domain-analysts only rather than having both domain-embedded analysts **and** domain-embedded IT professionals.

[3] We are skeptical that these two facts can be true at the same time. Our suspicion is that some respondents have not explicitly thought about version control as a distinct problem with purpose-built tooling that exists to solve it.

10 Conclusion

As far as we know, this is the first survey to investigate the current state of CPI Production Systems at National Statistics Organizations (NSO)s around the world. By introducing a generic set of concepts around system and team structure, we were able to derive some meaningful information about how CPI Production Systems and the teams that maintain them are organized. We were also able to characterize several important aspects of the teams maintaining these systems, including the tools and technologies they use, the age and update frequency of typical systems, number of collaborators, lead time, and challenges faced.

We begin by highlighting some noteworthy observations throughout the survey. Next, we provide some concrete and practical suggestions that we believe to be beneficial for both CPI Production Systems teams and teams maintaining Complex Analytical Systems more generally. Finally, we highlight future work that we believe will be productive.

10.1 Bottom Line Up Front

This section highlights some of the noteworthy results we found in our survey on CPI Production Systems.

10.1.1 System and Team Organization

- The two most common ways that system components are coupled across GSBPM steps are (1) to have one system span both the data ingestion step and the data processing step, and (2) to have one system span all 5 GSBPM steps.
- By far the most common team structure we found was comprised only of domain-embedded analysts. The second most common team structure we found was a mix of Corporate IT employees and domain-embedded analysts.
- Teams of domain-embedded analysts and domain-embedded IT professionals were not very common, suggesting that it is **not** common practice to embed IT staff within a business domain team. In other words, IT expertise tends to be centralized rather than embedded in business domain teams, which may contribute to increased communication overhead due to centralized IT staff lacking important business domain context.

- Using the "Representative System Group" question, we were able to elicit a high-level description of the system **and** team organization for a representative group of CPI Production Systems at the respondent's organization. While this system/team description was high level, it provided us with enough information to loosely group NSOs into 3 architecture categories (Monolithic, Hybrid, Modular) and 4 team categories (Stream-Aligned, IT-Only, Analyst-Only, Other Mix).
- IT-Only teams were the most likely to develop Modular Systems compared to the other 3 team types.
- The most common architecture type overall was Monolithic.
- Other Mix teams were much more likely to develop Monolithic systems compared to the other 3 team types[1].

10.1.2 Tools and Technologies

- A majority of respondents surveyed **don't use any kind of Version Control System at all**. The second most common answer was GitLab/GitHub, and the third most common version control strategy was to use "File Naming Conventions".
- Microsoft Excel was by far the most commonly used commercial software product in CPI Production System teams. The next most common product used was SAS, and the third most commonly used product is Microsoft Access.
- The most common tool used for project management was a shared Microsoft Excel workbook, while the second most common response was to not use any software for project management.
- The most common programming language used across all systems is SQL, while the second most frequently used language is tied between Python, R, and SAS.
- Among organizations with Monolithic representative systems, SQL is still the most commonly used language, with many other languages being reported. For example, Java, Python, Visual Basic Applications, R, Visual Basic, SAS, C#, and C++ are all being used by multiple organizations.
- Among organizations with Modular representative systems, R and SAS are tied for first place, with SQL in a close second. The only other responses with more than a couple of occurrences are "None" (i.e., no programming language is used), Python, and Visual Basic Applications.

[1]Note that there is a small sample size caveat with observations involving Other Mix teams. Had the sample size been larger, the observations we observed with this team type may not have been as extreme as what we observed in the survey.

- By far the most commonly used storage formats were (1) Database Management Systems (DBMS) and (2) file systems, with the former having a slightly higher occurrence.
- Very few respondents reported leveraging analytics-optimized file formats such as Apache Parquet, and very few respondents reported using Object Storage solutions such as Amazon S3 or Azure Data Lake Storage.

10.1.3 System Age and Updates

- Organizations with Monolithic representative systems are most likely to report a system age of 6-10 years or 11-20 years, while organizations with Modular representative systems are equally likely to report any of the system ages provided.
- IT-Only teams were much less likely to report a system age of "more than 20 years" compared to Stream Aligned teams and Analyst-Only teams.
- Other Mix teams were much more likely to report a system age of 6-10 years compared to any of the other options.
- Organizations with Monolithic representative systems were much more likely to report that the **majority of systems that are never updated** compared to organizations with Modular representative systems.
- Overall, the most common answers for how often the majority of systems are updated were (1) less than once per year, (2) never updated, and (3) once per year.
- Stream-Aligned teams were the most likely to report that the majority of systems are never updated, with all other team types reporting this answer multiple times[2]. Updating systems less than once per year was the most common answer across all team types.
- Other Mix teams didn't report any update frequency that was more frequent than every six months.

10.1.4 Number of People

- Most small changes require the participation of 2-3 individuals, with answers of "1 individual" and "4-6 individuals" also being somewhat common.
- Organizations with monolithic representative systems were slightly more likely to report a smaller number of individuals participating in **small changes** compared to organizations with modular representative systems.

[2]We suspect that this is slightly biased by the fact that the majority of Stream-Aligned teams in our sample are comprised of domain-analysts only rather than having both domain-embedded analysts **and** domain-embedded IT professionals.

- Organizations with monolithic representative systems were most likely to require participation of 4-6 individuals or 2-3 individuals for **large changes**, whereas organizations with modular representative systems were most likely to report requiring participation of 2-3 individuals. For both architecture types, there were a roughly equal number of answers requiring 7-9 individuals or more for large changes.

10.1.5 Lead Times

- The most common responses overall were lead times of "within 1 week" or "within 1 day" for **small changes**. However, a non-trivial fraction of respondents reported lead times of "within 1 month" or "within 3 months" for small changes.
- There was no meaningful difference between architecture types or team types for lead times on small changes.
- Organizations with monolithic representative systems were more likely to report lead times of "within 1 year" or "more than 1 year" for **large changes** compared to organizations with modular representative systems. A few respondents from organizations with monolithic representative systems reported that large changes were "too complex" or "the system can't be modified", whereas zero respondents from organizations with modular representative systems reported either of these answers.
- The most common lead time for **large changes** by far among organizations with stream-aligned teams was "within 1 month", whereas IT-Only teams were almost equally likely to report "within 1 week", "within 1 month", "within 3 months", or "more than 1 year".
- Other Mix teams reported the longest lead times for **large changes**, with every lead time reported being "within 1 month" or longer.
- Across all team types, "within 1 year" and "more than 1 year" were somewhat common answers to the question about lead times for **large changes**.

10.1.6 Alternative Data Usage

- Just under two thirds of respondents report not using alternative data at all.
- Of those respondents that do use alternative data, the majority of respondents report that "less than 10%" or "between 10% and 30%" of their CPI is comprised of alternative data by expenditure weight.
- Of those respondents that don't use alternative data, almost three quarters of them report that they would like to use alternative data.

- The most commonly cited challenges with respect to alternative data adoption are (1) lack of data provider cooperation, (2) insufficient capacity, and (3) insufficient skills to work with alternative data.

10.1.7 Overall Challenges Faced

- The most commonly cited challenge by far was "lack of staff", followed by "lack of skills maintaining complex systems" and "system interactions" in second and third place.
- Tied for fourth place are (1) managing complexity within a system (e.g., maintaining large quantities of code) and (2) verifying that a system behaves correctly.
- Almost noone cited "version control" as a challenge they faced, despite the majority of respondents indicating that they do not use any kind of version control solution or file naming conventions[3].

10.2 Practical Suggestions

Based on our survey results, we have several concrete and practical suggestions that may help the teams responsible for CPI Production Systems to manage complexity and reduce maintenance burden for these systems. While this guidance is targeted at the audience of this survey, our suspicion is that other teams managing similar Complex Analytical Systems may benefit from applying a number of the suggestions in this section.

> **!** Important
>
> All of these suggestions are guidelines based on correlational evidence and general industry knowledge.
> We are **not** claiming any strict cause-and-effect relationships from this survey alone. Rather, our goal is to encourage readers to think critically about these suggestions and to pursue suggestions that resonate based on their own experiences.

[3] We are skeptical that these two facts can be true at the same time. Our suspicion is that some respondents have not explicitly thought about version control as a distinct problem with purpose-built tooling that exists to solve it.

Table 10.1: Practical Takeaways from Survey

Suggestion	Explanation
Think explicitly about system boundaries.	There is some evidence in our survey that monolithic architectures are associated with certain outcomes such as (1) longer lead times for large changes and (2) never updating systems. Are two unrelated concerns implemented in the same system? Would it be easier to maintain your codebase if these concerns were split into two independent components?
Think explicitly about data interchange **between** systems.	If an important piece of data is exchanged between two or more systems, take the time to properly document important properties of the data such as the columns available, the data types, and the semantics of the data. This will make it easier for data consumers to use this data. One way to formalize this data exchange is through a standard format such as Data Contract Specification.
Embed technical expertise in business domain teams.	There is some evidence in our survey that "Other Mix" teams underperformed the other team mixes on several outcomes. Effectively developing and maintaining CPI Production Systems requires a team of individuals with **both** strong domain knowledge and strong technical skills. Whether technical expertise is embedded by directly employing IT professionals **within** the business domain team, or by upskilling domain-embedded analysts to improve their technical skills, our belief is that improving the technical capacity of business domain teams who own CPI Production Systems will lead to a number of improved business outcomes.
Adopt Git and GitLab or GitHub as a Version Control System for code, configuration, and documentation.	The cognitive load of understanding CPI Production Systems is already high enough without manually keeping track of version control and code integration from multiple collaborators. Allow a purpose-built tool handle the burden of revision control.

Suggestion	Explanation
Leverage analytics optimized file formats like Apache Parquet	Adopting a columnar file format like Parquet along with a handful of best practices makes it feasible to work with larger than memory datasets on a single machine. For example, if you only need to use 3 out of 50 columns of a table for a particular task, you can load tens of millions of records into memory on a modest commodity computer[4]. You can then work with this data using an industry standard data manipulation library such as Python's Pandas, Apache Arrow, or R Data Frames to name a few.
Where it's feasible, implement Complex Analytical Systems as pipelines with one-way data flows and idempotent operations.	If you can model your flow of change as directed acyclic graph (DAG) where nodes represent the state of data and edges between nodes represent idempotent operations, it is possible to significantly reduce the complexity of state management in your Complex Analytical System.A practical example of this concept is a data processing workflow that involves an upsert operation to update records[5]. If a batch of data contains records that are not in the original table, they will be inserted into the table, otherwise they will be updated in the table. What makes this operation **idempotent** is the fact that the same batch of data can be upserted to the table multiple times and the final result will be the same as if this operation happened only once.

[4]When working with this quantity of data, it is important to pay attention to the data types of each column and to choose the most parsimonious data types. For example, if 3 columns can be correctly represented with boolean, 16-bit integer, and 32-bit integer data types, there are significant memory savings to be gained by casting the variables to these types upfront.

[5]`UPSERT` is a portmanteau of the common `INSERT` and `UPDATE` database operations.

Suggestion	Explanation
Practice updating systems more frequently.	In software engineering, there is a commonly used performance metric called **Deployment Frequency** (Forsgren, Humble, and Kim (2018)), which measures how frequently code is deployed to production. The closest analog to this concept in the world of CPI Production Systems is what we've been referring to as "Update Frequency" throughout this report. The rationale for deployment frequency is that it's better to deploy many small and safe changes regularly than to deploy big risky changes infrequently. There are limits to how far this can be taken with CPI Production Systems, however, our belief is that frequently updating and testing systems with small changes leads to fewer unforeseen issues when it comes time to release a change for production[6].
Ensure your CPI Production System can operate in a separate development environment.	When dealing with complex systems, it is critical to have a safe environment where teams can "move fast and break things" without any risk to the production version of the system[7]. This is in contrast to **only** having the live production system to perform tests on, in which case one needs to be absolutely certain that a change won't irreparably break the live production system. In extreme cases, this proposition is so risky that the production system is never updated at all.

10.3 Future Work

We believe there are many areas of future work on this topic, but for brevity, we attempt to summarize them into two main categories below.

First, as mentioned at the beginning of this report, teams maintaining Complex Analytical Systems often struggle with many aspects of managing system complexity.

[6] In general, the Consumer Price Index is a non-revisable product, meaning that it is not straightforward to "roll back" a change in the same way that would be possible in other settings. Because of this, there is a certain level of due-diligence required for large system changes, meaning there are some limits on how frequently CPI Production Systems can be updated.

[7] Note that this can be as simple as having a `production` folder and a `development` folder on a network file system and scoping activities to the appropriate folder. More complex separations of development, testing, and production environments are possible, but we encourage readers to start simple if this is a new concept.

What is noteworthy is that none of these system complexity challenges are unsolved problems. As we note several times throughout our report, these problems have been examined extensively in other disciplines such as software engineering, and satisfactory solutions to these problems have often existed for decades.

Moreover, there is an abundance of freely available online material to teach these best practices, and there are often open source software products that address many of these challenges as well. There are even initiatives such as Reproducible Analytical Pipelines (RAP) (NHS (2017)) which actively curate the most relevant software engineering concepts for the kind of work required by Complex Analytical Systems.

Despite all of this, at the time of producing this report (April, 2025), effectively managing Complex Analytical Systems seems to be far from a solved problem. To this end, we think there is significant value in understanding the reasons this gap is so difficult to bridge. Importantly, is there a communication gap that can be bridged by more effectively connecting business domain analysts with the relevant content, or is the issue related to cognitive load or some other constraint limiting the bandwidth of these business domain teams?

Second, are there improvements that can be made to both (1) system architecture and (2) team organization for Complex Analytical Systems in order to improve business outcomes? For example, to what extent has Conway's Law impacted the architecture of Complex Analytical Systems such as the CPI Production Systems studied in this report? Would different team structures and team interaction modes lead to more effective system architectures?

Our survey provides some evidence that certain findings from the world of software engineering regarding team organization and system architecture may indeed be applicable to Complex Analytical Systems. However, the evidence from this survey alone is insufficient to make any sweeping generalizations about Complex Analytical Systems as a whole.

To this end, we believe there is significant business value to be gained by further investigating these topics in the context of other business domains maintaining Complex Analytical Systems[8].

[8]For example, if there is evidence that certain team structures, team interaction modes, and system architectures are more effective than others, then organizations that adopt the improved team topologies and system architectures may see reduced maintenance costs and faster delivery times, among other benefits.

References

Dehghani, Z. 2022. *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly. https://books.google.ca/books?id=M5J5zgEACAAJ.

Evans, E. 2004. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley. https://books.google.ca/books?id=7dlaMs0SECsC.

Ford, N., W. M. Richards, P. J. Sadalage, and Z. Dehghani. 2021. *Software Architecture: The Hard Parts : Modern Trade-Off Analysis for Distributed Architectures*. O'Reilly. https://books.google.ca/books?id=sWNozgEACAAJ.

Forsgren, N., J. Humble, and G. Kim. 2018. *Accelerate: The Science Behind DevOps : Building and Scaling High Performing Technology Organizations*. G - Reference,information and Interdisciplinary Subjects Series. IT Revolution. https://books.google.ca/books?id=85XHAQAACAAJ.

NHS. 2017. "Reproducible Analytical Pipelines (RAP)." https://nhsdigital.github.io/rap-community-of-practice/.

Richards, M., and N. Ford. 2020. *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media, Incorporated. https://books.google.ca/books?id=_pNdwgEACAAJ.

Skelton, M., M. Pais, and R. Malan. 2019. *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. G - Reference,information and Interdisciplinary Subjects Series. IT Revolution. https://books.google.ca/books?id=oFdRuAEACAAJ.