
Integrated Knowledge Management (IKM) Volume 3

Version 1 - Last Updated 11/21/2023

Table of Contents

I. IKM Overview	1
1. IKM Overview	3
1.1. Motivation for IKM	3
1.2. The Problem IKM Addresses	4
1.3. Goals of IKM	5
1.4. Terminology Knowledge Architecture (Tinkar) Overview	6
1.5. IKM Capabilities	7
1.6. IKM Challenges	7
1.6.1. Federal Regulation	7
1.6.2. Stakeholder Incentives	7
1.6.3. Coordination Across Standards	8
1.7. References	8
II. Current State of IKM	9
2. Summary of Findings: Technologies that Support Harmonization of Terminology Stan- dards	11
2.1. Introduction	11
2.1.1. Findings	11
2.1.2. Change Management	12
2.1.3. Dependency Management	13
2.1.4. User Interface (UI) and Experience (UX)	14
2.1.5. Quality Assurance (QA)	15
2.2. Conclusion	16
2.3. References	16
3. Current KM Technologies, Industry Coding Standards, and KM Hosting Options	18
3.1. Introduction to IKM	18
3.2. IKM Playbook Aims	18
3.3. IKM Playbook	18
3.3.1. Installer	18
3.3.2. Development Operations (DevOps)	19
3.3.3. Architecture	21
3.3.4. Classifier	22
3.3.5. Query and Retrieval	23
3.3.6. Import and Export	24
3.3.7. Editing	25
3.3.8. Persona	26
3.3.9. Contribution Model	27
3.3.10. Mobile Application	29
3.3.11. Human Centered Design	29
3.4. IKM Playbook Discussion	30
3.5. Ethical and Practical Considerations in Localized Artificial Intelligence (AI) and Machine Learning (ML) Development	30
3.5.1. Concerns About Localized AI/ML Models	30
3.5.2. Ethical Considerations and Concerns with Localized AI/ML Models	31
3.5.3. Recommendations for Ethical Practices	32
3.5.4. Recommendations and Guidelines for Small Vendors Developing Local- ized AI/ML Models	32
III. Development of IKM	35
4. Framework Considerations for Knowledge Management (KM)	37
4.1. Framework Considerations Purpose	37
4.2. Introduction to KM Processes	37
4.3. Current State Analysis	37

4.4. Methodology	37
4.4.1. Stakeholder Interviews	37
4.4.2. Requirements, Architecture, and Documentation Review	37
4.4.3. Code Review	38
4.5. Findings	38
4.5.1. Requirements Review	38
4.5.2. Architecture and Documentation Review	38
4.5.3. Code Review	38
4.5.4. Duplication	39
4.5.5. Layers and Coupling	40
4.6. Technology Choice Framework	40
4.7. Code Maturity	40
4.7.1. Documentation and Style Guidelines	40
4.7.2. Code Reviews	40
4.7.3. Immutability	43
4.7.4. How to Write Code Review Comments	45
4.7.5. Testing and the Cost of Defects	46
4.8. Automation	52
4.8.1. Continuous Integration (CI)	52
4.8.2. Continuous Deployment (CD)	54
4.8.3. Packaging and Distribution	54
4.8.4. Expected Changes to CI/CD	55
4.9. Scalability	55
4.10. License	55
4.11. Security	55
4.12. Privacy Needs	56

List of Figures

4.1. Unit Test Code Coverage Report for Komet Using JaCoCo	39
4.2. Unit Test Code Coverage Report for Tinkar using JaCoCo	39
4.3. Example of Package/Namespace Comments	44
4.4. Example of Class Comments	44
4.5. Example of Method/Function Comments	45
4.6. Example of Inline Comment	45
4.7. Relative Cost to Fix Bugs, Based on Time of Detection	46
4.8. Monetary and Time Costs of Testing Types	47
4.9. TDD Cycle Diagram	48
4.10. PseudocodeforCreatingMyArrayList	49
4.11. Simplest Code for Creating the MyArrayList Application	49
4.12. Example of Behavior Driven Development	51
4.13. Framework for Continuous Integration	52
4.14. Framework for Continuous Deployment	54

Part I. IKM Overview

Table of Contents

1. IKM Overview	3
1.1. Motivation for IKM	3
1.2. The Problem IKM Addresses	4
1.3. Goals of IKM	5
1.4. TermINology Knowledge Architecture (Tinkar) Overview	6
1.5. IKM Capabilities	7
1.6. IKM Challenges	7
1.6.1. Federal Regulation	7
1.6.2. Stakeholder Incentives	7
1.6.3. Coordination Across Standards	8
1.7. References	8

1. IKM Overview

1.1. Motivation for IKM

Knowledge management (KM) is an important process organizations use to identify, organize, and disseminate information and is providing greater value and insights into data with the emergence of technologies and advanced analytics, such as artificial intelligence (AI) and machine learning (ML). Individuals and organizations alike are continuously generating data which may sit in siloed databases but, with the appropriate tools, can be aggregated across platforms, analyzed, and utilized. However, the flow of data through systems can lead to a myriad of issues including loss of meaning, inability to aggregate data, misinterpretation, and more. Achieving coordinated and standardized ways to integrate data across systems is an important tenant of integrated knowledge management (IKM).

These issues are no different for healthcare and laboratory data. The introduction of the clinical information systems, including electronic health record (EHR) systems and laboratory information systems (LIS), in the 1960's revolutionized the way an individual's health data was stored. Health data went from static data recorded in paper records to dynamic, encoded knowledge stored in informational systems and introduced the ability to aggregate and examine these data points to generate new ideas and solve problems. Removing ourselves from the idealized state of full knowledge integration, the current landscape is composed of a range of EHR platforms, LISs, users, standards organizations, and agencies, all of which employ their own methods for collecting, recording, and using healthcare data. The healthcare system's transformation into a digitalized health information system was quickly adopted but several core high reliability organization (HRO) principles were excluded, including ensuring the system was interoperable, data was of the highest quality, and patient safety was prioritized.

Several organizations have emerged to address these issues through industry coding standards and implementation guidance. However, these have fallen short of ensuring high quality data with a common representation or standardization across the healthcare ecosystem. Standards including Systematized Nomenclature of Medicine Clinical Terms® (SNOMED CT®), Logical Observations Identifiers Names and Codes® (LOINC®), National Library of Medicine's codification of the U.S. pharmacopeia (RxNorm) and others exist to address the variation in healthcare data and resulting inefficiencies by working with terminology, standards, and meaning. Each organization follows a set of rules that govern the standards and add structure. However, even the best-meaning user may interpret the use of standards differently, highlighting the need to move beyond the reliance on standards alone. The integration of the above-mentioned organizations, users, and standards are important to create an interoperable and high-quality healthcare system.

Currently, implementers must know how to work with all the existing standards and be familiar with their respective rules and formats. Through this body of work, we are advancing a reference demonstration of an IKM environment that can be used by various stakeholders in the laboratory ecosystem to easily view, manage, and integrate existing knowledge standards (e.g., SNOMED CT®, LOINC®) in a singular location, ultimately allowing users to identify relationships between disparate knowledge sets. This work will help set the foundation for a common model that would eventually help promote collaboration across standards-development organizations, vendors, and provider-organizations, improving interoperability of laboratory data across the entire ecosystem. Through IKM, all standards work together under one data model with tooling to work across standards and reduce redundancies. IKM can benefit the healthcare ecosystem and society by improving patient outcomes, allowing meaningful data analyses, and increasing interoperability.

1.2. The Problem IKM Addresses

In healthcare, organizations that help achieve standardized health data are referred to as Standards Development Organizations (SDOs). SDOs exist in every industry, but in health Information Technology (IT) the following are key examples:

SNOMED CT®	A national standard, comprehensive and precise multilingual health terminology globally [1]
LOINC®	Logical Observation Identifiers Names and Codes terminology: facilitate the exchange and pooling of clinical or laboratory results for clinical care, outcomes management, claims attachment, and research by providing a set of universal codes and names. LOINC® codes allow users to merge clinical results from many sources into one database for patient care, clinical research, or management [2]
RxNorm	Standard that provides normalized names for clinical drugs and links its names to many of the drug vocabularies commonly used in pharmacy management and drug interaction software [3]
Fast Healthcare Interoperability Resources (FHIR)	Application programming interface (API) focused standard that enables electronic health data to be quickly and efficiently exchanged [4]

The Office of the National Coordinator for Health Information Technology (ONC) currently oversees and promotes SDOs and supports the adoption and application of health data standards in the U.S. healthcare space. The structure and meaning of data must be maintained to achieve full data integrity and a truly interoperable system. FHIR is an example of a standard that works to maintain the structure of data as it flows through systems, while terminology organizations including SNOMED CT®, RxNorm, and LOINC® work to maintain the meaning of data. With the many existing SDOs and agencies pushing their adoption, issues such as overlapping coding standards, loss of meaning, and partial representation continue to exist.

While medical terminology standards exist, the way they record, manage, and represent terminologies differs, leading to redundant and duplicative ways to reference the same concept. This leads to variations in clinical data exchange across the ecosystem as organizations use inconsistent and varied concept representation. Interoperability specifications aim to force these terminologies to adhere to predefined concepts, code systems, and reusable value sets. Although there are standardized codes and formats, there are many ways to use these standards.

To address the current issues with disparate standards, a common model and IKM capabilities are needed to support the integration and management of clinical terminology and local concepts to support increased data quality and interoperable clinical information. [5] IKM and implementation of a common model can support the integration of disparate standards, provide version control, and support extensions that otherwise do not exist. A common model for knowledge standards addresses the differences in management and structure across disparate terminology standards, local concepts, and code lists/value sets. Employing the architectural design principle of Separation of Concerns (SoC), a system can be divided into sections and each section can be addressed and allowed to evolve irrespective of other sections. In IKM and its knowledge architecture, the foundational layer provides the common elements of interoperability including object identity, versioning, modularity, and knowledge representation. In this foundational architecture layer, we find the shared module system where version and configuration management provide the foundation for interoperability and the common model.

A common model will unify disparate standards and address the following points when designing an interoperable healthcare system:

1. **Version Control:** changes to published terminology artifacts will be documented and historic versions maintained if the need for rollback arises.
2. **Extensions:** Improved processes around extensions to terminology that conform to the requirements of the standards the extension is based on. The ability to implement terminology extensions and support additional content

1.3. Goals of IKM

One of the goals of this body of work is to develop both a common model and the necessary tooling to support the implementation of the common model and IKM capabilities. Our work aims to improve patient care and safety through interoperable, high-quality data, and accurate transmission within and between healthcare systems. Improved data quality will cascade to other refinements within the healthcare ecosystem, such as improved research capabilities, identification of real-world data trends, and real-time public health information. A standardized way to represent medical terminology must be developed to ensure new concepts are represented in a uniform and repeatable way and support the conversion of existing terminology standards into a common model while maintaining meaning, associated concepts, semantics, hierarchies, and other critical information and relationships.

To support IKM, an implementation of a common model for knowledge standards must support a variety of functionalities across all knowledge standards and frameworks, including:

- **Import** – The data model must be able to pull and visualize data from LOINC®, SNOMED CT®, and other terminology standards into the model's data elements. Displaying the disparate terminology standards in a single place will allow users and authors to compare commonalities and differences between standards, identify equivalent concepts, and understand how concepts relate to one another.
- **Edit** – The model must be able to convert pulled data into the standardized format while maintaining meaning and any associated concepts. Editing will support the identification of unnecessary or redundant concepts caused by unstandardized formatting of concepts across terminology standards. Editing within the model will also support a uniform way to safely and reliably update, evolve, and version content over time as new concepts, relationships, and information are uncovered.
- **Authoring** – The model must support the development of new concepts using a standardized and repeatable format to maintain lossless capture and transmission of data (including identification and inclusion of related concepts), incorporate new concepts such as Long coronavirus disease 2019 (COVID-19), and meet standardized terminology formatting.

Tooling is needed to support IKM and the common model. The tooling will serve as an environment to view all existing standards and how they are related. Examples of functionality include:

- Concept, semantic, hierarchy, and pattern display and navigation
- Options to view additional details of various concepts
- Clear and concise versioning information
- Equivalent concept lists

1.4. Terminology Knowledge Architecture (Tinkar) Overview

Today's healthcare system uses certain information systems to capture and manage clinical statements and terminologies. While multiple information systems and terminology standards exist, these standards represent and format equivalent concepts, semantics, and models in different ways and manage new concept release cycles and versioning updates asynchronously. Due to this disparate terminology representation, identifying equivalent concepts, transmitting data without losing key details or meaning, and ensuring the quality of data is a challenge. Through this body of work, we are aiming to improve IKM capabilities and advancing an existing Health Level Seven International (HL7) project within the Vocabulary Working Group known as the 'Guidelines for a Standardized Terminology Knowledge base', also known as Terminology Knowledge Architecture (Tinkar). Tinkar is intended to provide an architecture that delivers integrated terminology to the healthcare enterprise and its information systems. Tinkar addresses the differences in management and structure across reference terminology, and with local concepts and code lists/value sets. [5]

Tinkar is a self-describing and machine processed knowledge architecture that allows for effective and scalable querying of data, concepts, and information. Using locally defined concepts, Tinkar can integrate various terminology standards into the common model by accommodating, managing, and defining granular components and their respective data sources to support interoperability of semantics. Having the ability to manage complex and individual elements of each standard then allows continual process improvement, including rapid extension development. Edits to concepts or newly authored concepts in Tinkar can be tested and previewed before deployment to a live environment and historic versioning of meta data is perpetually maintained to support iterative authoring, development, and versioning. [5]

Tinkar aims to improve the quality of patient care across the healthcare enterprise by creating a cross-cutting and common model that can integrate other terminology standards and use a repeatable and standardized process to convert disparate clinical statements and terminologies into high-quality, interoperable, and exportable clinical data. The four main problems that Tinkar and a common model can address are:

1. Suboptimal ability to identify equivalent concepts within and between terminology standards
 - Concepts like "Feels Feverish" and "Feels Hot/Feverish" are both concepts in SNOMED CT® but are not recognized as equivalent within SNOMED CT®. Tinkar's standardized process would support the identification of these findings as equivalent and create a relationship between the concepts.
2. No standardized way to represent newly identified concepts in real-time
 - Terminology standards currently lack the ability to rapidly adapt and deploy pertinent concepts in response to emerging phenomenon, such as COVID-19. Tinkar would allow for a standardized and repeatable process to rapidly develop and version concepts in response to real-time healthcare developments.
3. Accidental use of incorrect or less accurate concepts due to a lack of harmonization between terminology standards
 - The lack of standardized representation of concepts between standards leads to unnecessary confusion, redundant and overlapping concepts, and challenges identifying the most accurate concept when transmitting data between systems and terminology standards. Tinkar works to reduce this unnecessary complication with clear versioning hierarchies that allow for precise identification and representations of appropriate terminologies between standards.
4. Asynchronous versioning that does not clearly convey detailed descriptions of changes

- Versioning within terminology standards is not currently well documented or clearly conveyed to users and versioning between terminology standards is done asynchronously. Tinkar will support transparent, detailed, standardized and rapid versioning that focuses on improvements for users.

1.5. IKM Capabilities

This work aims to demonstrate how Tinkar architecture can optimize and implement IKM capabilities and IKM principles through the development of a reference implementation through collaboration with federal agencies and the Systemic Harmonization and Interoperability Enhancement for Laboratory Data (SHIELD) collaborative community. SHIELD aims to support increased patient safety through improved interoperability, transmission, and quality of laboratory data between and within healthcare systems. [6]

1.6. IKM Challenges

While IKM promises a revolutionary way to improve the quality and interoperability of healthcare data in the name of improved patient safety and patient outcomes, achieving a truly integrated ecosystem will pose a variety of challenges. Creating an IKM architecture is itself a major undertaking, but will require a deep, fundamental understanding and a detailed approach to address the following challenges. These challenges are not exhaustive and will evolve as this work continues.

1.6.1. Federal Regulation

As with any new or emerging technology, a variety of Federal agencies will need to play a major part in the implementation of an IKM system. Not only is their review and approval of the proposed system a necessity before implementation with patient facing healthcare systems, but the federal government will also need to develop incentives to encourage the private sector to participate and meet the minimum requirements needed to support IKM. Due to the inherent complexity and holistic approach to data, IKM will require multiple federal agencies to support and drive participation, like:

- The Centers for Medicare and Medicaid Services (CMS) and Office of the National Coordinator for Health IT (ONC) will need to outline minimum data requirements for orders to ensure the participating entities are capturing standardized data elements for standardized questions that come from a well-known and widely available source of truth.
- The FDA will need to develop an authorization process that requires new and existing terminology standard organizations, healthcare systems, product developers, and more to comply with pertinent standards prior to incorporation into the IKM platform.
- The Centers for Disease Control and Prevention (CDC) must provide basic standards and easily accessible guidance to enforce standardized data collection, aggregation, and transmission for the various healthcare systems within the IKM platform.

It is critical to coordinate with federal agencies as early as possible to gather and incorporate their feedback, provide extensive information detailing our work, the intended goals, and benefits, and collaborate on requirements to support improved data quality and patient safety. A detailed stakeholder engagement plan that details timelines, stakeholder dependencies, and required actions is required to efficiently develop these requirements.

1.6.2. Stakeholder Incentives

Federal agencies will need to develop minimum requirements and incentives to encourage higher education, private sector, and other stakeholders to participate. Meeting the minimum requirements for data

collection and data transmission will require changes to the user interface (UI) and user experience (UX) that standardize data entry requirements, representation of data elements, and transmission of data on the front and back end of healthcare and laboratory systems. These could be extensive changes depending on the age and current data collection of the systems. Before hospitals and laboratories commit both financial and personnel resources to perform gap assessments, identify areas of improvement, implement changes, train staff and users, and sustain updates, regulators and implementers will need to use statistical analysis and evidence-based discussions to highlight the current problems with data and how these problems directly impact or relate to each stakeholder on an individualized level. Demonstrations showing how improvements to patient safety, optimization of data collection, transmission, and querying, and the IKM environment with statistics analysis will address these current state issues and use financial incentives to encourage stakeholders to achieve the optimized future state we are working towards.

1.6.3. Coordination Across Standards

A defining feature of IKM is working to harmonize various terminology standards using a common language to standardize data capture while maintain meaning and associated concepts throughout transmission and storage. This requires working with a variety of terminology standards that asynchronously represent, update, manage, and version concepts. The IKM architecture will require a robust and standardized process to track changes across terminology standards and version Tinkar concepts appropriately in a repeatable way. Changes will need to be made in real time and will need to communicate the intricacies of updates, test the validity and functionality of changes, and maintain Status, Time, Author, Module, and Path (STAMP) principles. Changes to concepts may occur concurrently as terminology standards release updates and will need to be tested and validated using the Tinkar authoring environment.

1.7. References

1. National Library of Medicine. Overview of SNOMED CT [Internet]. Nih.gov. U.S. National Library of Medicine; 2016. Available from: https://www.nlm.nih.gov/healthit/snomedct/snomed_overview.html
2. LOINC — The freely available standard for identifying health measurements, observations, and documents. [Internet]. Loinc.org. 2019. Available from: <https://loinc.org/>
3. National Library of Medicine. RxNorm [Internet]. www.nlm.nih.gov. 2020. Available from: <https://www.nlm.nih.gov/research/umls/rxnorm/index.html>
4. FHIR Fact Sheets | HealthIT.gov [Internet]. www.healthit.gov. Available from: <https://www.healthit.gov/topic/standards-technology/standards/fhir-fact-sheets>
5. HL7 Logical Model: Standardized Terminology Knowledgebase, Release 1. Terminology Knowledge Architecture Informative Ballot. HL7 Vocabulary Work Group. August 2021. https://www.hl7.org/documentcenter/private/standards/HL7_LM_TERM_KB_R1_INFORM_2021AUG_FINAL.pdf
6. Health C for D and R. Systemic Harmonization and Interoperability Enhancement for Laboratory Data (SHIELD). FDA [Internet]. 2023 Mar 24 [cited 2023 May 31]; Available from: <https://www.fda.gov/medical-devices/diagnostic-data-program/systemic-harmonization-and-interoperability-enhancement-laboratory-data-shield>

Part II. Current State of IKM

Table of Contents

2. Summary of Findings: Technologies that Support Harmonization of Terminology Standards	11
2.1. Introduction	11
2.1.1. Findings	11
2.1.2. Change Management	12
2.1.3. Dependency Management	13
2.1.4. User Interface (UI) and Experience (UX)	14
2.1.5. Quality Assurance (QA)	15
2.2. Conclusion	16
2.3. References	16
3. Current KM Technologies, Industry Coding Standards, and KM Hosting Options	18
3.1. Introduction to IKM	18
3.2. IKM Playbook Aims	18
3.3. IKM Playbook	18
3.3.1. Installer	18
3.3.2. Development Operations (DevOps)	19
3.3.3. Architecture	21
3.3.4. Classifier	22
3.3.5. Query and Retrieval	23
3.3.6. Import and Export	24
3.3.7. Editing	25
3.3.8. Persona	26
3.3.9. Contribution Model	27
3.3.10. Mobile Application	29
3.3.11. Human Centered Design	29
3.4. IKM Playbook Discussion	30
3.5. Ethical and Practical Considerations in Localized Artificial Intelligence (AI) and Machine Learning (ML) Development	30
3.5.1. Concerns About Localized AI/ML Models	30
3.5.2. Ethical Considerations and Concerns with Localized AI/ML Models	31
3.5.3. Recommendations for Ethical Practices	32
3.5.4. Recommendations and Guidelines for Small Vendors Developing Localized AI/ML Models	32

2. Summary of Findings: Technologies that Support Harmonization of Terminology Standards

2.1. Introduction

Health IT standards are critical for representing and sharing health data in a large health ecosystem. Multiple users with roles in various fields can produce and access health data, compelling the need for a common way to format the data standards. This is particularly important for data encoded in EHRs and LISs. Standards, such as SNOMED CT®, LOINC®, and RxNorm, provide a common language for certain types of health information to be represented and understood. However, multiple industry knowledge standards exist in a variety of environments and need to be harmonized to ensure data interoperability. An IKM platform is needed to harmonize these standards, ultimately generating high quality data for downstream use.

Knowledge standards are currently managed by different organization and stakeholder groups. These standards are often promulgated by the organizations to advance their use; however, the evolution of these standards occur in silos, compounding the complexity of standard harmonization. Clinical data and concepts can be represented by multiple standards, leading to overlap, and impeding the ability to recognize semantic equivalence between the standards. A core component of standard harmonization is the ability to maintain both structure and meaning to ensure data integrity. Interoperability challenges arise when only one of these tenants is observed. These challenges lead to misinterpretations, inefficiencies, and declines in patient safety.

An IKM Platform is needed to harmonize standards and promote data interoperability across the ecosystem. In an integrated laboratory data ecosystem, knowledge can freely flow between systems in a smooth exchange. Benefits of this exchange include improved collaboration, incorporation of advanced analytical tools, and true data interoperability. The need for a knowledge management platform and integrated laboratory data system is the same; coordinating knowledge management practices across organizations and systems to achieve a safe and effective healthcare system.

2.1.1. Findings

While HRO principles and IKM serve as a well-structured architecture to support the harmonization and standardization of disparate clinical terminology standards, there are key advancements that must be made and technologies that must be utilized to optimize this work.

2.1.1.1. Data Management

Data management is the safe and efficient storage, management, and use of collected data to support informed decision making and improve outcomes for individuals and organizations. This is a critical area of the healthcare enterprise that must undergo advancements to support optimized knowledge standard harmonization, improved patient safety and outcomes, and enhanced data quality. [1] While data management covers a wide range of processes and functions, this work focuses on three key areas:

1. **Ease of Access** – While an increasing number of healthcare organizations rely on IT systems to store, manage, and access data collected from a variety of sources, the ability to identify, access, and then utilize that data in an impactful and meaningful way is an ongoing challenge. This work aims to improve both the author's and user's ability to efficiently capture data from across the healthcare ecosystem and to effectively analyze that data for actionable decision making.

2. **Integrating Disparate Data Sources** – As healthcare organizations capture data from different sources, it is critical that disparate terminology standards and data representation can be integrated into a common model. While various terminology standards exist, the formatting and representation of clinical data across standards can vary significantly, leading to issues with data interoperability and suboptimal concept representation between standards and systems. Using a standardized and repeatable process, our team will work to ensure the common model can integrate and represent data from a variety of sources, including well established standards and newly emerging systems.
3. **Data Integrity** – Patient care and patient safety are innately tied to the quality and integrity of their healthcare data. Currently, as data is transferred within and between healthcare systems, including EHRs, LISs, and pharmaceutical systems, it can often lose meaning and pertinent information regarding relationships and associated concepts. This work will develop a common model that will use a standardized process to ensure data integrated from disparate data sources maintains meaning and all pertinent information throughout data transmission, storage, and use. Tools such as universally unique identifiers (UUID) will support Tinkar with the identification of semantically equivalent concepts and support optimal data representation across terminology standards.

Tinkar

Tinkar is a self-describing knowledge architecture intended to represent other terminology standards in a human readable and machine processable format. Tinkar supports improvements to data management by integrating terminology standards into a common model that efficiently captures granular details and data sources of each standard for rapid and scalable data querying and analysis while maintaining meaning. Versioning is another important component of maintaining data integrity. Tinkar will maintain a detailed version history to clearly convey how concepts have changed over time and will allow authors to preview or test edits to concepts before pushing to a live environment to maintain appropriate concept representations and relationships. [2] As new healthcare systems or terminology standards emerge, the data they represent will be “Tinkarized” and integrated into Tinkar and the IKM architecture.

Protocol Buffers

Protocol buffers are an agnostic and unopinionated way to represent structured and packeted data that can easily be pushed or pulled to different IT systems. This system and standard-neutral representation of data allows a wider range of systems and coding languages to read the same data than traditional methods and supports rapid data transmission, long-term storage, forward and backward compatibility, and improved data querying. Protocol buffers will allow Tinkar to import an encoded concept from one system’s terminology standard, store and manage that data, and then transmit that data with maintained data integrity to another system to display the concept in the disparate, local terminology standard. [3]

UUID version 5

UUID version 5 uses a 128-bit label to idempotently represent data in a completely unique and unambiguous way. Idempotent data representation means that identical seed values or concepts will be labeled with the same UUID, supporting identification of semantically equivalent concepts within and between terminology standards and optimized access to data. Additionally, by labeling each concept, semantic, and pattern within Tinkar with a UUID, our team is able to integrate a variety of terminology standards while avoiding duplicative or redundant concepts, preventing accidental and inappropriate labeling of a concept or component, and harmonizing internal representation of knowledge standards. [4]

2.1.2. Change Management

Change management is simply an approach to identify, implement, and support necessary organizational changes. Despite each terminology standard and healthcare system employing some form of change management, they are implemented asynchronously and lack detailed tracking and communication of changes. Even within a terminology standard the change management for models, definitions, and structure can

vary in time and frequency. Supporting harmonization of knowledge standards through IKM requires a comprehensive change management strategy to track, identify, and reflect independent changes across systems to appropriately represent updated concepts in Tinkar based on source code changes.

Change Sets

Change sets are a group of changes that are accompanied by meta-information that clearly and concisely describe the packaged changes as they are committed to a live environment. Change sets support optimized change management by providing granular details of each change and enables revisions to be made if concept changes need corrections. [5]

STAMP

STAMP is a syntax used to represent current and previous terminology assets (Concepts, patterns, and semantics) in Tinkar using the following five requirements:

1. **Status** – The status is an indication of whether an asset is active or inactive.
2. **Time** – The time and time zone at which a change was recorded.
3. **Author** – The author of a change, identified using the appropriate granularity.
4. **Module** – The module is an organizational label for the larger asset to which a component belongs, such as code, system, or edition.
5. **Path** – The path is a production component of the previously defined organization, such as development, testing, staging, or deployment. [2]

STAMP is not only used to version Tinkar data, but it is also a requirement for all Tinkar assets, existing and new. Ensuring all data follows this syntax supports optimal change management and versioning practices, like maintaining detailed versioning history by marking concepts as active or inactive rather than deleting prior versions, identifying authors of each version to know who and when concepts were changed, and testing and previewing changes before they are pushed to a live environment. [2]

Key Value Store

A key value store, or key value database, stores data as a value and assigns a key to that data, which is then used to track, retrieve, and update or change the data associated with a given key. When used in parallel with STAMP, a key value store would maintain the key for a certain value/data whose status changed from active to inactive or underwent other changes to update author and time, further supporting Tinkar's detailed versioning and change management best practices. This type of database would also support rapid and efficient querying of healthcare data to support improved patient outcomes, public health trends, and policy making. [6]

2.1.3. Dependency Management

Harmonization of multiple terminology standards and healthcare systems using IKM will require an extensive approach to dependency management, the process and approach to managing dependencies between people, process, and systems. With Tinkar, outputs to one system can often originate from a wide range of terminology standards or systems, which leads to multiple dependencies. It is not only important for our team to identify dependencies, but we must also categorize them to understand the direction, frequency, and type of dependency and to optimally prioritize and mitigate potential issues caused by these relationships. For example, Laboratory Interoperability Data Repository (LIDR) has a dependency on both SNOMED CT® and LOINC® and when SNOMED CT® or LOINC® are updated, validation must be performed in a comprehensive manner to determine if LIDR is correctly using the current interpretation of said dependent standards (SNOMED CT® and LOINC®). As works continues towards IKM, it is critical

that technologies are selected and implemented that will support the management of these dependencies to reduce patient safety issues and unnecessary complications. [7]

Maven

Maven is a project management tool with dependency management features for complex, multi-module projects. Maven compliments STAMP and key value stores because it allows project owners to name artifacts and specify the version and build stage (development, testing, deployment, etc.) involved in a dependency. Maven can also import dependencies from other projects, supporting continual process improvement and integration of new systems and terminology standards. [8]

Nexus

Nexus is a repository used to manage components, binaries, and build artifacts across software. In Komet, which is described below, this allows the developer to organize and control the versions of data that are released, stored, and made available to users. Nexus provides enterprise control through centralization, storage, development support, and scale. [9]

Origin data

An optimized representation of a source knowledge standard is generated to simplify the process of putting the standard's data into a common model. This data will leverage Maven and Nexus to manage dependency complexity of systems using Tinkar data, since individual knowledge standards have both physical and logical models for how they are constructed, packaged, and distributed. An important aspect in managing the dependency of harmonized standards is organizing the source data (raw) into structures. Origin data will repackage the source terminology data into a consistent and readable format, which alleviates the need to write individual ways to parse and read through the different variations of file structures. Origin data also allows the addition of data that is not found in the source terminology data (e.g., provenance, dependencies [LIDR depends on LOINC® and SNOMED®]).

Starter data

Tinkar starter data acts as the foundational dataset. It conforms to the Tinkar data model and is the minimum viable dataset required for Komet usability. The Tinkar starter data consists of a Tinkar concept model hierarchy which defines origin and destination (a.k.a. parent/child) concept relationships, as well as Tinkar patterns which define the structure and usage for how Tinkar concepts may be represented. The Tinkar data model is extendable and allows for the importation and harmonization of other terminology standards such as SNOMED CT® and LOINC® and enables a lossless transformation of knowledge standards to the Tinkar format. Starter data will leverage Maven and Nexus to manage dependency complexity of systems using Tinkar data.

2.1.4. User Interface (UI) and Experience (UX)

Harmonizing self-sustaining and contained knowledge standards requires a robust and tailored UI/UX design. A strong UI/UX interface meets the needs of the user, is intuitive to use, and provides an environment for tackling complex terminological overlaps between various standards which is needed for properly harmonized knowledge. Without a good user interface, work to harmonize standards cannot be done. Good UI/UX design provides users a visible interface to work with different coding standards. Users come from a variety of backgrounds including laboratory technicians, clinicians, researchers, regulatory personnels, and others and cannot all be expected to be proficient coders who can readily access and use the backend software/coding language. Good UI/UX is built off human centric design principles, is tailored to different roles and needs, and provides components that give the user the ability to create, edit, and retire knowledge. Tinkar core is the harmonization which lives in the backend, but UI/UX provides an interface that allows users to interact with the logic and harmonize between different coding standards. UI/UX can also incorporate features such as multiple languages and dialects, which allow for a user to have a tailored ex-

perience and can be focused on a particular regional preference for descriptions and other text information to emphasize different features which may prompt the desired response or use from the specific user.

Komet

Komet serves as the reference implementation for IKM capabilities. Komet will combine developer tools into one graphical UI that will allow developers to write, edit, debug, compile code, and automate tasks for software development. [10] Komet will be open-source, meaning the code being developed is open to the public and available for all to edit and distribute. This decentralized production model is a movement in the software community and has led to major advancements in code and is often cheaper, flexible, and has more permanency than private or proprietary software. [11] Komet has automated features such as a text editor, build automation, and debugger, and can support additional features like refactoring, code search, and continuous integration and continuous deployment (CI/CD) tools. Komet supports plug-ins and extensions for developers to customize workflows and needs. [12] All standard harmonization will be facilitated within Komet.

Coordinates

Tinkar is the data model for knowledge harmonization and IKM. The ability to query these Tinkar representations, such as comparing active and inactive components (representations) across time, is achieved through coordinates. [2] Coordinates include concepts such as STAMP, language, dialect, clinical domains, and more. STAMP is the most basic of the coordinates, and through which all content is filtered. Examples of STAMP coordinates include the most recent version, a set of data from several versions, and all active components only.

- **Language Coordinates** control the terms to be displayed based on a language/dialect and list of synonyms.
- **Logic Coordinates** identify results from Description Logic Classifiers and different output versions over time.
- **Navigation Coordinates** allow users to view and search certain concepts, with examples including stated versus inferred relationships for SNOMED CT®, and concepts included/excluded for certain domains.

The ability to search, display, and navigate concepts and semantics requires the ability to calculate combinations of content based on different coordinates. The current Tinkar reference model is expressive enough to represent any terminology and future iterations will require harmonized implementation guides for each terminology. [2]

2.1.5. Quality Assurance (QA)

Quality assurance (QA) is important when harmonizing knowledge standards and is critical when assessing if a software meets the users' requirements. Constraints can be used to perform general QA on content that is represented in a Tinkar implementation and is part of business requirements. Constraints in Tinkar are used to represent standard terminology artifacts and are used to ensure terminologies represented in an implementation are consistent when queried and displayed, and ultimately ensure the integration is high quality and reliable for patient care. Furthermore, to ensure the quality of items that are transformed and integrated together a type of classifier can be created to check for issues or errors in Tinkar data.

ELK Classifier

Komet's ability to use a description logic classifier to detect equivalences and errors between concepts is important when harmonizing knowledge standards. In particular, the ELK Classifier, a description-logic classifier, is an ontology reasoner that supports Web Ontology Language (OWL) 2 Existential Logic (EL) profile and provides a quick reasoning engine for OWL EL because current features and reasoning tasks

are limited (although this is fine for SNOMED CT®). This classifier reasons over Tinkar data and distinguishes if the asserted facts on concepts (via semantics) are true. [13]

2.2. Conclusion

Disparate terminology standards and healthcare systems continue to plague today's healthcare ecosystem with patient safety challenges associated with data quality and interoperability. To improve patient outcomes and support informed policy and public health decision making, comprehensive and holistic IKM is needed. Harmonizing terminology standards will ensure data can be transmitted within and between healthcare systems while maintaining meaning, associated concepts, and other pertinent information. This paper outlines various technologies and tools that are needed to support critical aspects of IKM.

As our team continues to work towards the harmonization of terminology standards through Tinkar and IKM, it is critical that we continue to refine and incorporate the various components of data, change, and dependency management and quality assurance mentioned above. The components outlined in this paper are not exhaustive and will evolve as our team continues with our work and as new information emerges.

2.3. References

1. What is Data Management? [Internet]. [cited 2023 Jun 1]. Available from: <https://www.oracle.com/database/what-is-data-management/>
2. HL7 Logical Model: Standardized Terminology Knowledgebase, Release 1. Terminology Knowledge Architecture Informative Ballot. HL7 Vocabulary Work Group. August 2021. https://www.hl7.org/documentcenter/private/standards/HL7_LM_TERM_KB_R1_INFORM_2021AUG_FINAL.pdf
3. Protocol Buffers - Overview [Internet]. [cited 2023 Jun 1]. Available from: <https://protobuf.dev/overview/>
4. Universally Unique Identifier [Internet]. Wikimedia Foundation; 2023 [cited 2023 Jun 1]. Available from: [https://en.wikipedia.org/wiki/Universally_unique_identifier#Versions_3_and_5_\(namespace_name-based\)](https://en.wikipedia.org/wiki/Universally_unique_identifier#Versions_3_and_5_(namespace_name-based))
5. Changeset [Internet]. Wikimedia Foundation; 2022 [cited 2023 Jun 1]. Available from: <https://en.wikipedia.org/wiki/Changeset>
6. What is a Key-Value Database? [Internet]. [cited 2023 Jun 1]. Available from: <https://www.mongodb.com/databases/key-value-database>
7. Rogers P. Dependency Management Techniques [Internet]. A Path Less Taken; 2022 [cited 2023 Jun 1]. Available from: <https://medium.com/agile-outside-the-box/dependency-management-techniques-187f888a6aad>
8. Porter B, Laugstol T, Marbaise KH. Apache Maven Project - Introduction to the Dependency Mechanism [Internet]. [cited 2023 Jun 1]. Available from: <https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html>
9. Sonatype Nexus Repository - Binary and Artifact Management | Sonatype [Internet]. www.sonatype.com. Available from: <https://www.sonatype.com/products/sonatype-nexus-repository>
10. Okeke F. The 12 best IDEs for programming [Internet]. TechRepublic. 2022. Available from: <https://www.techrepublic.com/article/best-ide-software/>
11. Redhat. What is open source? [Internet]. Redhat.com. 2019. Available from: <https://www.redhat.com/en/topics/open-source/what-is-open-source>

12. RedHat. What is an IDE? [Internet]. Redhat.com. 2019. Available from: <https://www.redhat.com/en/topics/middleware/what-is-ide>

13. OwlFeatures [Internet]. GitHub. [cited 2023 Jun 2]. Available from: <https://github.com/liveontologies/elk-reasoner/wiki/OwlFeatures>

3. Current KM Technologies, Industry Coding Standards, and KM Hosting Options

3.1. Introduction to IKM

The healthcare system in America has undergone numerous changes throughout the past decades, most notably in the way it records, retains, and transfers medical records. Within these records, clinical laboratory data, including orders, results, and interpretations are among the most important types of data. This information is used in a variety of settings including clinical care, public health, and even the development of drugs and medical devices.

With the evolution of healthcare data has come the evolution of how it's recorded. Today health care and laboratory systems use terminology standards such as LOINC® and SNOMED CT®. The long-term challenge to using these standards is that laboratory data is often recorded differently within and between institutions, which impacts reliability, interoperability, accuracy, and, more critically, patient safety. In addition to different standards and interoperability challenges, version control of the knowledge standards and equivalence identification amongst standards pose potential patient safety issues. Data interoperability tooling and change management is needed to implement a standardized model for health IT standards and address differences in the management and structure of coding standards, local concepts, and codes lists/ value sets across the industry.

3.2. IKM Playbook Aims

A product playbook is a guide for helping developers, product owners, and managers to build products faster, with fewer bugs and better user experience. It is a collection of high-level user stories and elaborated acceptance criteria. This playbook specifically showcases the Komet contributed application and identifies necessary functionality and capabilities to establish a collaborative, robust, and highly reliable Knowledge Management Platform (KMP). It will serve as a guide for the development and implementation of the Komet contributed application and other technical components (e.g., Tinkar-core, Web Application), while also suggesting specific technologies and products that could be used in the development process.

The overarching objectives of this playbook are to:

1. provide a high-level overview of key functional areas of a reference implementation for IKM capabilities.
2. establish user-centric scenarios to describe necessary KMP features and functionalities.

3.3. IKM Playbook

Each sub-section below breaks down key areas that emphasize core capabilities for the KMP through the articulation of user stories and a list of descriptive acceptance criteria.

3.3.1. Installer

As a Komet contributed application user, I want to install the latest version of the Komet contributed application easily across various types of computers and devices, so that I can utilize the latest capabilities and features for working with Tinkar-based knowledge data:

- Given an end-user wanting to install the Komet contributed application on their Windows desktop, when they click the installation file (e.g., .msi), then the installer should straightforwardly install the desktop component of the Komet contributed application.
- Given an end-user wanting to install the Komet contributed application on their macOS desktop, when they click the installation file (e.g., .pkg), then the installer should straightforwardly install the desktop component of the Komet contributed application.
- Given an end-user wanting to install the Komet contributed application on their Linux desktop, when they click the installation file (e.g., .deb), then the installer should straightforwardly install the desktop component of the Komet contributed application.
- Given an end-user wanting to install the mobile Komet contributed application on their iPhone Operating System (iOS) device, when they navigate to the iOS App Store, search for Komet, and select “install”, then the App Store should seamlessly install the iOS component of the Komet contributed application.
- Given an end-user wanting to install the mobile Komet contributed application on their Android device, when they navigate to the Google Play Store, search for Komet, and select “install”, then the Google Play Store should seamlessly install the Android component of the Komet contributed application.
- Given an end-user running the Komet desktop installer, when they select an option to install a sample Tinkar data set, then the installation wizard should copy the appropriate data set into <Home>/Solor directory.
- Given an end-user running the Komet desktop installer, when they select an option to install custom Tinkar data sets, then the installation wizard should copy the appropriate data sets into <Home>/Solor directory.
- Given an end-user running the Komet desktop installer, when they select an option to install custom remote Tinkar data sets, then the installation wizard should download and copy the appropriate data sets into <Home>/Solor directory.
- Given an end-user running the Komet desktop installer, when they select an option to utilize a remote Tinkar datastore, then the installation wizard should configure Komet to work with the user selected datastore.
- Given an end-user running the Komet mobile installer (iOS, Android), when they select an option to install sample Tinkar data set, then the installation wizard should copy the appropriate data set into the correct mobile app directory.
- Given an end-user running the Komet mobile installer (iOS, Android), when they select an option to custom remote Tinkar data set, then the installation wizard should copy the appropriate data set into the correct mobile app directory.
- Given an end-user running the Komet mobile installer (iOS, Android), when they select an option to utilize a remote Tinkar datastore, then the installation wizard should configure Komet to work with the user selected datastore.

3.3.2. Development Operations (DevOps)

As a Komet contributed application, Tinkar-core, and Web Application developer, I want to follow industry leading Development Operations (DevOps) best practices, so that the projects I contribute to are able to safely manage and reduce risk incurred when I commit source code changes:

- Given a software developer committing code changes to the Komet contributed application, Tinkar-core, or the Web Application repository, when GitHub receives the push notification for the commit, then

a subsequent event hook should be sent to Jenkins to start appropriate continuous integration and continuous delivery activities.

- Given GitHub sending an event hook on a commit to Jenkins, when Jenkins receives the hook event, then the server should create the appropriate pipeline(s) necessary to build, test, quality assure, and publish appropriate software artifacts.
- Given Jenkins receiving a commit event hook from GitHub, when Jenkins builds the appropriate pipeline integrated with quality assurance tooling, then there should be configurable thresholds or gateways for each quality assurance tooling output to cancel the continuous integration and continuous delivery process.
- Given Jenkins receiving a commit event hook from GitHub, when Jenkins creates a build pipeline, then the pipeline should integrate and utilize SpotBugs quality assurance scanning as a quality control gateway.
- Given Jenkins receiving a commit event hook from GitHub, when Jenkins build pipeline for the source code is created, then the pipeline should integrate and utilize SonarQube quality assurance scanning as a quality control gateway.
- Given Jenkins receiving a commit event hook from GitHub, when Jenkins build pipeline for the source code is created, then the pipeline should integrate and utilize Black Duck quality assurance scanning as a quality control gateway.
- Given Jenkins receiving a commit event hook from GitHub, when Jenkins build pipeline for the source code is created, then the pipeline should integrate and utilize a code coverage analysis tool as quality control gateway.

As a Komet contributed application, Tinkar-core, and Web Application developer, I want to make available the latest artifacts and binaries necessary for the rest of my team to develop their features on, so that all developers are using the latest and most available artifacts and dependencies:

- Given a successful execution of a build pipeline, when Jenkins determines all quality control gateways have passed, then Jenkins will publish the newest build of the source code as artifacts to a private artifact repository.

As an open-source contributor to the Komet contributed application, Tinkar-core, and Web Application projects, I want to be able to recommend my source code changes to the main developer team, so that I can help improve the code base in an open, transparent, and collaborative way:

- Given a software developer committing code changes to the Komet contributed application, Tinkar-core, or the Web Application repository, when GitHub receives a pull request notification, then a subsequent event hook should be sent to Jenkins to start appropriate continuous integration and continuous delivery activities.

As a developer of health IT systems, I want to easily be able to use Tinkar based projects and solutions within my own implementations, so that my health IT systems become more highly reliable and improve patient safety:

- Given a successful execution of a build pipeline, when Jenkins determines all quality control gateways have passed, then Jenkins will publish the newest build of the source code as artifacts to a public Maven artifact repository.
- Given a successful execution of a build pipeline, when Jenkins determines all quality control gateways have passed, then Jenkins will subsequently generate a new installer artifact to be published to publicly available repository.

- Given a successful execution of a build pipeline, when Jenkins determines all quality control gateways have passed, then Jenkins should update the appropriate project badges within the source code versioning system to reflect quality aspects (e.g., code coverage, build status).

As a DocBook contributor to the IKM volumes, I want to follow industry leading DevOps best practices, so that the content I contribute to the volumes are safely managed:

- Given a content contributor committing code changes to the IKM project, when GitHub receives the push notification for the commit, then a subsequent event hook should be sent to Jenkins to start appropriate continuous integration and continuous delivery activities.
- Given a software repository sending an event hook on a commit to Jenkins, when Jenkins receives the hook event, then the server should create the appropriate pipeline(s) necessary to build, test, quality assure, and publish appropriate software artifacts.
- Given a successful execution of a build pipeline, when Jenkins determines all quality control gateways have passed, then Jenkins will publish the newest build of the IKM volumes as a PDF to a public repository.

As an open-source user of the Tinkar-based projects, I want to be able to report any bugs and/or issues identified while using the Komet contributed application and the Web Application, so that I can improve the maturity and stability of all Tinkar-based project's source code:

- Given after a bug and/or issue has been identified by an end-user, when the user navigates to the correct project's Jira page, then they should have access and the appropriate privileges to enter in a bug and/or issue ticket.

3.3.3. Architecture

As a software architect, I want to create and outline necessary Tinkar-based application programmable interfaces, design specifications documents, and other software artifacts, so that current and future development of the Komet contributed application, Tinkar-core, and the Web Application align with both the technical and functional vision of the KMP:

- Given there are several Tinkar-based software solutions, when a developer is designing new features and capabilities to meet current and emerging needs, then there needs to be an enterprise level design document that conveys how each software component interacts and manages Tinkar data.
- Given that the Tinkar-based software is trying to establish a collaborative, contributed application to share and manage various types of knowledge, when a developer is designing new features and capabilities to meet current and emerging needs, then there needs to be a Contribution model design document that conveys how various Tinkar-based software components share Tinkar data and what supporting technologies are necessary to facilitate reliable merging and extending of Tinkar data created and maintained by multiple users.
- Given the abundance of popular data processing methodologies and frameworks within the software and data science industry, when a developer is designing new features and capabilities to meet current and emerging data processing needs, then there needs to be a Tinkar Processing Model design document that conveys what types of data processing paradigms are appropriate when working with Tinkar data in a KMP.
- Given the abundance of popular client server architectures and a focus on empowering clients to develop custom-tailored data queries, when a developer is designing new features and capabilities to meet current and emerging data processing needs, then there needs to be a GraphQL Integration Model design

document that conveys how GraphQL can reduce implementation complexity when querying based on Tinkar coordinates.

- Given the benefit of using Application Programming Interfaces (APIs) to standardize the implementation of various components within a complex software architecture, when a developer is designing new imports and exports for an arbitrary knowledge standard to be transformed into a Tinkar-based data representation, then there needs to be a Tinkar Extract Transform Load (ETL) API that safely constrains how a developer can implement an ETL algorithm on a knowledge standard.
- Given the complexity of the Tinkar data model and the impact that a particular datastore's design can have when trying to work with bespoke data models (such as Tinkar), when a developer is designing new features and capabilities to meet current and emerging data persistence needs, then there needs to be a Tinkar Datastore design document that outlines appropriate supporting data store technologies and describes in detail how to implementations using such technologies.
- Given the past development efforts on the KMP, when a developer is designing new features and capabilities to meet current and emerging business needs, then there needs to be an initial collection of documentation artifacts created that explain the current existing code base as it has evolved.
- Given the current and future development efforts on the KMP, when a developer is designing new features and capabilities to meet current and emerging business needs, then there needs to be incremental updates to all documentation artifacts that accurately explain the current code base.
- Given the recent improvements to Java's JavaDoc API, when a developer is designing new features and capabilities to meet current and emerging business needs, then they should utilize the latest JavaDoc technology to centralize developer documentation within the code base itself.

As an end-user of the KMP, I want to be able to access my Tinkar knowledge data through various formats and across multiple devices, so that I am not limited to just a desktop when working with my Tinkar data:

- Given the need for a user to access Tinkar data across multiple form factors, when a user wants to access Tinkar data, then there needs to be an implementation of a centralized Web Application that follows reactive web technologies to enable simple Tinkar data browsing and curation across various web enabled devices.
- Given the effort to document a design for the most appropriate way to query for Tinkar data given a client service architecture pattern, when a user wants to provide Tinkar data to their health IT systems, then there needs to be an implementation of GraphQL API that retrieves Tinkar data for an arbitrary amount of clients.

As an end-user of the KMP, I want to be able to store my Tinkar data in the most optimal and appropriate format, so that my KMP environment will provide the best performance for my knowledge management needs:

- Given the effort to document a design for the most appropriate datastore technology for Tinkar data, when a user wants to provide Tinkar data to their health IT systems, then there needs to be an implementation of a "best of breed" datastore that is specifically tailored to store Tinkar data.
- Given the effort to customize a datastore to specifically work with the Tinkar data structure, when a user wants to implement the best database for their KMP, then there needs to be an analysis and review of how to optimize a select number of datastores for Tinkar data.

3.3.4. Classifier

As a knowledge engineer, I want to be able to utilize a classifier and/or reasoner on my Tinkar knowledge data, so that all my curation efforts are validated in a safe, efficient, and reliable manner:

- Given the current implementation of Snorocket in Tinkar-core, when a knowledge engineer makes small edits to their Tinkar data, then the classifier should only perform classification in an incremental fashion on the concepts effected by the edits.
- Given the need to provide incremental classification on Tinkar data, when a user wants to classify their changes made to Tinkar data, then Tinkar-core and/or Komet need to provide an ELK classifier implementation.
- Given the complexity of integrating an ELK classifier into the current Tinkar-core and Komet code bases, when a user wants to classify their changes made to Tinkar data, then there needs to be a formal validation process to reduce the risk of improper implementation and incorrect classification results.
- Given the need to provide efficient classification on Tinkar data and the effort to migrate from Snorocket to ELK, when a user wants to classify their changes made to Tinkar data, then there needs to be a review of how best to extend the default ELK classifier implementation.
- Given the limited resources available in the informatics community that understand the ELK classifier, when the developers improve on the ELK classifier through their Tinkar integration efforts, then there needs to be tangible contributions back to the originating ELK Classifier reference implementation.
- Given the amount of information that is outputted from the ELK classifier, when a knowledge engineer runs the classifier on Tinkar data, then all outputs from the classifier need to be based on cohesive and succinct user interface and user experience design.
- Given the criticality of the results outputted from the ELK classifier, when a knowledge engineer runs the classifier on Tinkar data, then Komet must provide quality assurance capabilities to utilize classifier outputs to further refine and/or correct Tinkar data edits.
- Given the amount of information that is displayed and contained within the Komet contributed application, when a knowledge engineer finishes edits on Tinkar components, then the process to run the classifier should be minimally intrusive and coherent to the authoring workflow and experience as designed within Komet.

3.3.5. Query and Retrieval

As a Knowledge Engineer, I want to be able to construct and execute both simple and advanced type queries on my Tinkar data, so that I can meet current and unforeseen health IT systems knowledge data needs:

- Given the need for a simple information retrieval of Tinkar data, when a user performs a search, then the search functionality should be implemented using Apache Lucene.
- Given the need for advanced information retrieval of Tinkar data, when a user performs a search, then the search functionality must implement a For, Let, Where, Order by, Return (FLWOR)-based query methodology.
- Given the need for a user to quickly filter what information is viewed within the Komet contributed application or the Web Application, when a user modifies a Tinkar Coordinate property value, then the Tinkar data being displayed must be re-retrieved and reflect the recently changed Tinkar Coordinate.
- Given the need to have efficient searching of Tinkar data, when a user performs a search, then there needs to be an analysis and review of how to optimize both the simple Lucene and FLWOR-based searches.
- Given the novelty and customization that is involved in implementing both simple and advanced search capabilities within the KMP, when a developer wants to utilize search features within the KMP, then there needs to be sufficient API and JavaDoc documentation.

- Given the extensibility of Lucene search queries, when a user performs a simple search, then there should be documentation and resources available to explain the current capabilities of Lucene searching.
- Given complexity of the FLWOR query methodology, when a user performs an advanced search, then there should be documentation and resources available to explain the FLWOR query capabilities.
- Given the novelty of Tinkar Coordinates, when a user performs a coordinate filter, then there should be documentation and resources available to explain how Tinkar Coordinates are designed and provide computational-based information retrieval.

3.3.6. Import and Export

As a Knowledge Engineer, I want to be able to import meaningful and relevant knowledge standards into my KMP, so that I can continuously work to harmonize, curate, and improve how my health IT systems use various knowledge standards:

- Given the need to import SNOMED CT® into the KMP, when a user selects a release of SNOMED CT® within the Komet contributed application, then Komet should perform an ETL process that correctly transforms SNOMED CT® into Tinkar components.
- Given the need to import LOINC® into the KMP, when a user selects a release of LOINC® within the Komet contributed application, then Komet should perform an ETL process that correctly transforms LOINC® into Tinkar components.
- Given the need to import RxNorm into the KMP, when a user selects a release of RxNorm within the Komet contributed application, then Komet should perform an ETL process that correctly transforms RxNorm into Tinkar components.
- Given the need to import the SNOMED CT® LOINC® Collaboration into the KMP, when a user selects a release of SNOMED CT® LOINC® Collaboration within the Komet contributed application, then Komet should perform an ETL process that correctly transforms SNOMED CT® LOINC® Collaboration into Tinkar components.
- Given the need to import Global Substance Registration System (GSRS) data into the KMP, when a user selects a release of GSRS within the Komet contributed application, then Komet should perform an ETL process that correctly transforms GSRS data into Tinkar components.
- Given the need to continuously maintain an up-to-date representation of a particular knowledge standard within the KMP, when a user selects a knowledge standard release to import, then there should be an option to only import the “difference” between releases of said knowledge standard.
- Given the need to import knowledge standards that are maintained in a custom format, when a user selects the custom format within the Komet contributed application, then Komet should prompt the user with an interface that articulates how that format will follow the Tinkar ETL process.
- Given the need to export Tinkar data from the KMP as SNOMED CT® knowledge, when a user selects the SNOMED CT® export feature within the Komet contributed application, then Komet will transform all existing Tinkar data into an appropriate and valid SNOMED CT® structure and release format.
- Given the need to export Tinkar data from the KMP as LOINC® knowledge, when a user selects the LOINC® export feature within the Komet contributed application, then Komet will transform all existing Tinkar data into an appropriate and valid LOINC® structure and release format.
- Given the need to export Tinkar data from the KMP as RxNorm knowledge, when a user selects the RxNorm export feature within the Komet contributed application, then Komet will transform all existing Tinkar data into an appropriate and valid RxNorm structure and release format.

- Given the need to export Tinkar data from the KMP as GSRS knowledge, when a user selects the GSRS export feature within the Komet contributed application, then Komet will transform all existing Tinkar data into an appropriate and valid GSRS structure and release format.
- Given the need to export knowledge standards that are maintained in a custom format, when a user selects the custom format export feature within the Komet contributed application, then Komet should prompt the user with an interface that articulates how current Tinkar data and formatting will be transformed into the custom export format.

3.3.7. Editing

As a Knowledge Engineer, I want to be able to curate my Tinkar knowledge data in a transparent and highly reliable way, so that I can predict the impact my knowledge data has on patient care and health outcomes:

- Given Tinkar's data model and the importance of a unifying Concept component, when a user wants to create a new Concept, then Tinkar-core should utilize STAMP versioning to create a new Concept.
- Given Tinkar's data model and the importance of a unifying Concept component, when a user wants to update a Concept, then Tinkar-core should utilize STAMP versioning to modify an existing Concept's Time value.
- Given Tinkar's data model and the importance of a unifying Concept component, when a user wants to delete a Concept, then Tinkar-core should utilize STAMP versioning to modify an existing Concept's Status value.
- Given Tinkar's data model and the importance of a contextualizing Semantic component, when a user wants to create a new Semantic, then Tinkar-core should utilize STAMP versioning to create a new Semantic.
- Given Tinkar's data model and the importance of a contextualizing Semantic component, when a user wants to update a Semantic, then Tinkar-core should utilize STAMP versioning to modify an existing Semantic's fields and Time value.
- Given Tinkar's data model and the importance of a contextualizing Semantic component, when a user wants to delete a Semantic, then Tinkar-core should utilize STAMP versioning to modify an existing Semantic's Status value.
- Given Tinkar's data model and the importance of a definitional Pattern component, when a user wants to create a new Pattern, then Tinkar-core should utilize STAMP versioning to create a new Pattern.
- Given Tinkar's data model and the importance of a definitional Pattern component, when a user wants to update a Pattern, then Tinkar-core should utilize STAMP versioning to modify an existing Pattern's field definitions and Time value.
- Given Tinkar's data model and the importance of a definitional Pattern component, when a user wants to delete a Pattern, then Tinkar-core should utilize STAMP versioning to create a modify and existing Pattern's Status value.
- Given the variations in types of fields that a Semantic could possibly contain, when a user decides to edit primitive typed fields, then Komet's user interface should allow for in-line editing of values.
- Given the variations in types of fields that a Semantic could possibly contain, when user decides to edit Tree and/or Graph fields, then Komet should prompt the user with a graph and/or tree friendly interface to modify vertex values.

- Given the variations in types of fields that a Semantic could possibly contain, when user decides to edit EL++ (axioms) or description logic fields, then Komet should prompt the user with an EL++ compliant editing environment and/or interface.
- Given the natural workflow of having multiple authors collaborate on a single Tinkar component (Concept, Semantic, Pattern), when a different user modifies a Tinkar component, then that specific component should utilize STAMP versioning to capture a new version of the Author value.
- Given the harmonization capabilities when having multiple knowledge standards represented as reusable Tinkar components (Concept, Semantic, Pattern), when a user modifies a Tinkar component shared across multiple knowledge standards, then that specific component should utilize STAMP versioning to associate the component changes to the correct Module value.
- Given the branching ability of Tinkar's components, when a user wants to promote, change or merge a branch context of a component, then that specific component should utilize STAMP versioning to modify the Path value.

As a Knowledge Engineer, I want to be able to perform editing and curation tasks at scale, so that I can keep up with the volume and velocity of knowledge standard updates within my health IT enterprise:

- Given the integrated nature of Tinkar components and its ability to represent various knowledge standards, when a user edits a Tinkar component, then Komet should apply similar quality assurance rules as defined by the imported source knowledge standards.
- Given the integrated nature of Tinkar components and its ability to represent various knowledge standards, when a user edits a Tinkar component, then Komet should apply any configured quality assurance rules as defined by other users.
- Given the complexity of representing and curating several knowledge standards together as Tinkar components (Concept, Semantic, Pattern), when a user wants to intermittently save their curation progress (e.g., modifying semantic field values), then Komet should store all modifications as transactions until the user is ready to commit them as formal Tinkar component edits.
- Given the need to apply similar Tinkar component edits across more than one individual component, when a user selects multiple Tinkar components to apply modifications to, then Komet will process those modifications adherent to a batch process framework.
- Given the need to apply similar Tinkar component edits across more than one individual component within a data stream, when a user defines multiple Tinkar components to apply modifications to, then Komet will process those modifications adherent to a stream process framework.
- Given the need to provide real-time feedback when editing an individual concept based on configured quality assurance rules, when a user tries to edit a particular Tinkar component that contains a quality assurance rule, then Komet should instantly display feedback regarding the validity of the component's edit and adhere to a real-time process framework.

3.3.8. Persona

As a Knowledge Engineer, I want to be able to customize the layout, interfaces, and functionality of my KM environment, so that I can more easily accommodate various associated workflows when working with knowledge standards:

- Given the need to provide a customizable user experience with the Komet contributed application and/or the Web Application, when a developer is working on importing a new knowledge standard, then there should be a set of detailed persona design documents that describe what user interface components are configurable within the Komet contributed application and the Web Application.

- Given the need to customize the Komet contributed application and/or the Web Application, when a user selects create a persona, then the user interface should open a configuration window where the user can create a new persona to modify all aspects of the Komet contributed application and/or the Web Application.
- Given the need to customize the Komet contributed application and/or the Web Application, when a user selects update a persona, then the user interface should open a configuration window where the user can update an existing persona.
- Given the need to customize the Komet contributed application and/or the Web Application, when a user selects delete a persona, then the user interface should open a configuration window where the user can delete an existing persona.
- Given individual preferences of each Komet contributed application or Web Application user as to how they curate knowledge standards, when a user opens the persona configuration window and selects “Workflow”, then they should be presented with a customizable sequence of STAMP Path values that can “promote” a Tinkar component.
- Given the need to support various combinations of languages and dialects within the Komet contributed application and the Web Application, when a user opens the persona preferences window and selects “Language”, then there should be options that work to modify the “Language” coordinate associated with the persona (e.g., preferred language, preferred dialect, prioritized display of description [synonyms vs text vs fully qualified name]).
- Given the need to support various types of logical classifiers within the Komet contributed application and the Web Application, when a user opens the persona preferences window and selects “Logic”, then there should be options that work to modify the logic coordinate associated with the persona (e.g., filtered results based on classifier, filtered versions from classifiers).
- Given the need to support various ways to view and search for Tinkar components within the Komet contributed application and the Web Application, when a user opens the persona preferences window and selects “Navigation”, then there should be options that work to modify the navigation coordinate associated with the persona (e.g., stated vs inferred relationships, concept inclusion vs exclusion).
- Given the need to support various ways to view all Tinkar components within the Komet contributed application and the Web Application, when a user opens the persona preferences window and selects “STAMP”, then there should be options that work to modify the STAMP coordinate associated with the persona (e.g., most recent version, Set of data from several versions, all active components only).
- Given the need to show only specific or relevant tabs and panels within the Komet contributed application and the Web Application, when a user opens the persona preferences window and selects “Dynamic Layout”, then there should be options that limit, re-arrange, remove, and add any combination of panels, tabs, and windows to the Komet contributed application and Web Application interface associated with the persona.
- Given the need to integrate the Komet contributed application and the Web Application with other technologies and services that capture user preferences, when a user opens the persona preferences window and selects “3rdParty Integrations”, then there should be options to login and integrate the Komet contributed application and the Web Application with external preference services (e.g., GitHub).

3.3.9. Contribution Model

As a Systems Integrator, I want to be able to transmit and receive binary-formatted Tinkar data in a language and implementation agnostic context, so that I am not limited by programming languages, implementation frameworks, and other design choices when integrating Tinkar into my various health IT systems:

- Given the need for a language agnostic, framework neutral, and binary formatted Tinkar data representation, when developers are working to integrate Tinkar data into their health IT systems, they should be following a formalized Tinkar Protocol Buggers representation release format.
- Given the evolution of the Tinkar data standard as more and more knowledge standards are transformed into Tinkar representations, when a developer is integrating the Protocol Buffers release format into an existing health IT system, then there needs to be a strict adherence to Google's implementation best practices for safely evolving a Protocol Buffers implementation across modifications and versions.

As a Knowledge Engineer, I want to be able to collaborate and share my knowledge representation work with other Knowledge Engineers using a controlled and highly reliable contribution model, so that I can curate, evolve, and incorporate the good works and best practices of other KMP users:

- Given the modularity of Tinkar's data structure and its STAMP versioning, when a user wants to collaborate or share Tinkar data, then a process can be performed that calculates the difference between a preexisting version given coordinate and STAMP conditions which will produce a concise "diff" release of Tinkar data.
- Given the high probability of having conflicting changes with another user on the same Tinkar data, when a user wants to share their Tinkar data with another user and merge conflicts are detected, then the Komet contributed application and the Web Application will display a prompt that identifies the merge conflicts and offers an interface to perform reconciliation with all contributing users.
- Given the need to ensure that shared Tinkar knowledge data still meets your organizational or personal quality assurance process, when a user shares with or syncs their Tinkar data with other users, then there should be processes in place that allow users to define quality assurance checks against merged Tinkar data.
- Given the need to support both the localization of knowledge standards (collaboratively) and the national/international release of knowledge standards, when a knowledge engineer is finished editing a Tinkar component, then the Komet contributed application and/or the Web Application should be able to generate a valid release format of a given imported knowledge standard.
- Given the benefits of collaborative knowledge representation engineering, when a user starts up the Komet contributed application and/or the Web Application, then there should be a configurable feature to "auto sync" or pull Tinkar data from other users.
- Given the polarization of interpretation of knowledge extensions, when a user decides to sync their Tinkar data (pull from another repository) with another user, then they should have the ability to control which users they auto pull Tinkar data from in a fine-grained way.

As a Knowledge Engineer, I want to focus on establishing a collaborative workflow using the KMP with my team, so that we can curate and improve the knowledge standards used in our health IT enterprise:

- Given the need for me, as a team lead, to assign review of certain problematic Tinkar knowledge data, when I identify the Tinkar component that needs review, then the Komet contributed application and/or the Web Application will enable me to assign a member of my team (user) to resolve the component issue.
- Given the ability for me to assign component review for my teammates, when I do assign work to a specific user, then the Komet contributed application and the Web Application will provide that user with the appropriate alerts and notifications that there has been an assignment.
- Given the potential resolution by my team member of an assigned problematic Tinkar component, when my team member believes they have resolved the issue, then the Komet contributed application and/or the Web Application will provide them with the ability to request approval for proposed component resolution.

- Given that a request for approval for a proposed component resolution has been sent to me, when I perform a review of the resolutions, then I would like to either promote the resolution or fix to a higher “branch” or path (e.g., development to production).
- Given the ability for my team members to be creative in establishing their own personas to improve their workflows, when I ask my teammate to share their persona configuration, then the Komet contributed application and/or the Web Application should be able to load my team members’ persona configurations.

As a Knowledge Engineer, I want to be able to restrict some of my Tinkar data collaboration, so that I can maintain my organization’s proprietary interests:

- Given that some of my Tinkar data is proprietary, when I go to sync my Tinkar data with other users, then I should have the ability to constrain or isolate which Tinkar components get shared with other users.
- Given that some of my Tinkar data is proprietary, when I login to the Komet contributed application and/or the Web Application, then there should be a robust user access control infrastructure to grant proper authentication and authorization to users.

3.3.10. Mobile Application

As a Knowledge Consumer, I want to be able to access and view Tinkar data on my various devices, so that I can view Tinkar data from any location:

- Given the need for an iOS application for Tinkar data, when a developer wants to build a Tinkar iOS app integration, then there should be some iOS Tinkar Application design document that conveys the ability to integrate critical parts of Tinkar into the iOS application framework.
- Given the fact I own an Apple device, when I want to view Tinkar data, then I should be able to see some basic form of Tinkar data in an iOS application.
- Given the need for an Android application for Tinkar data, when a developer wants to build a Tinkar Android app integration, then there should be some Android Tinkar Application design document that conveys the ability to integrate critical parts of Tinkar into the Android application framework.
- Given the fact I own an Android device, when I want to view Tinkar data, then I should be able to see some basic form of Tinkar data in an Android application.

3.3.11. Human Centered Design

As a Knowledge Consumer, I want to have customized interfaces and experiences when using Tinkar data, the Komet contributed application, the Mobile Application, and/or the Web Application, so that I can best use Tinkar to solve my business needs:

- Given the uniqueness of my business needs as an end-user, when a designer starts to understand the necessary capabilities, then there should be a landscape analysis conducted to better understand similar types of needs from similar types of end-users within a specific domain.
- Given the uniqueness of my business needs as an end-user, when a designer starts to understand the necessary capabilities, then there should be stakeholder interviews to understand specific functionality from a selected types of users.
- Given the uniqueness of my business needs as an end-user, when a designer starts to understand the necessary capabilities, then there should be a heuristic analysis conducted to better understand the current state of the user interface and experience regarding the Komet contributed application and/or the Web Application.

3.4. IKM Playbook Discussion

This Product Playbook is detailed but not exhaustive. For this playbook to provide a continuous impact on all aspects of design, development, and evolution of Komet and KMP over time, there must be regular updates. This playbook will serve as a living document that provides durable value to the various teams working on the Komet contributed application. There has been a great history of valiant efforts, brilliant design, and profound ideas that have shaped the current iteration of Komet as we know it today. This Playbook serves as both a historical record and a future roadmap to make the Komet contributed application and the KMP real.

3.5. Ethical and Practical Considerations in Localized Artificial Intelligence (AI) and Machine Learning (ML) Development

3.5.1. Concerns About Localized AI/ML Models

While localized development of AI and ML offers customization and optimization benefits, the associated risks and challenges necessitate careful consideration. Without adequate safeguards and oversight, localized development can lead to biased, unreliable, and inequitable AI/ML algorithms, compromising patient care and outcomes. The concerns associated with localized development highlighted below document the need for a more standardized, equitable, and generalizable approach to developing AI/ML algorithms in healthcare.

1. Bias Risk

- Concern; There is a risk of introducing or worsening biases when developing algorithms locally especially if the dataset used is not diverse enough.
- Impact; This could result in predictions and unreliable outcomes, for groups potentially leading to suboptimal care for these patients.

2. Limited Applicability

- Concern; Algorithms developed and validated on localized data may not perform well when applied in healthcare settings or patient populations.
- Impact; The lack of applicability can limit the usefulness and effectiveness of the algorithm in healthcare contexts.

3. Inconsistency in Healthcare

- Concern; Different healthcare facilities may develop their localized algorithms resulting in inconsistent sepsis detection and care approaches.
- Impact; This inconsistency can cause confusion among healthcare providers. Compromise the quality and standardization of care across settings.

4. Regulatory Hurdles

- Concern; Regulatory bodies may face challenges when evaluating and approving versions of localized algorithms.

- Impact; The variability introduced by localized development adds complexity to the approval process and oversight.

5. Challenges with Maintenance and Updates

- Concern; Managing, maintaining and updating localized algorithms can be resource intensive and complex.
- Impact; This complexity can lead to difficulties, with version control (as in fact mentioned by Shawn in the document he shared on the teams thread), updates and continuous improvement of the algorithms.

6. Ethical and Equity Concerns

- Issue; The development and implementation of localized algorithms raise concerns regarding fairness, equity and ethical considerations.
- Effect; If not properly regulated localized algorithms could worsen healthcare disparities and give rise to concerns.

7. Lack of Standardization

- Issue; The presence of localized algorithms may lead to a lack of standardization, in the detection and treatment practices for sepsis.
- Effect; This absence of standardization can impede the integration and compatibility of AI/ML tools, across healthcare systems.

3.5.2. Ethical Considerations and Concerns with Localized AI/ML Models

Allowing smaller sepsis AI device manufacturers to pursue localized development raises several ethical considerations and requires careful oversight. Ethical considerations include;

1. Equity and Fairness

- Concern: Smaller manufacturers might have access to limited datasets, which may not be representative of diverse patient populations. This limitation can lead to the development of biased algorithms that do not perform equitably across different groups.
- Ethical Question: Is it ethical to deploy algorithms that might not provide equitable benefits to all patient groups due to limitations in data diversity and representativeness?

2. Quality and Safety

- Concern: Smaller manufacturers might lack the resources to conduct extensive validation and testing of their algorithms, potentially compromising the quality and safety of their devices.
- Ethical Question: Can we ensure that devices developed by smaller manufacturers meet the necessary quality and safety standards to protect patient health and well-being?

3. Transparency and Accountability

- Concern: There might be variations in the level of transparency and accountability among smaller manufacturers, raising concerns about the ethical deployment and use of their devices.

- Ethical Question: How can we ensure that smaller manufacturers adhere to ethical standards of transparency and accountability in the development and deployment of their devices?

4. Regulatory Compliance

- Concern: Smaller manufacturers might face challenges in navigating and complying with regulatory requirements, potentially leading to non-compliance and associated risks.
- Ethical Question: Is it ethical to allow the deployment of devices that might not fully comply with regulatory standards and requirements due to limitations in resources and expertise?

5. Access and Affordability

- Concern: Devices developed by smaller manufacturers might have limited availability and accessibility, potentially exacerbating healthcare disparities.
- Ethical Question: How can we ensure that devices developed by smaller manufacturers are accessible and affordable to all patients who might benefit from them?

3.5.3. Recommendations for Ethical Practices

While smaller sepsis AI device manufacturers pursuing localized development raises several ethical consideration, with the appropriate support, resources, and guidelines, it is possible to address the ethical concerns associated with localized development by smaller manufacturers that support innovation and diversity in the field.

1. **Ethical Oversight;** It is important to establish mechanisms for oversight in order to thoroughly review and monitor the development and deployment of devices by manufacturers (if not already there).
2. **Support and Resources;** Smaller manufacturers should be provided with support and resources to ensure their adherence to standards and regulatory requirements.
3. **Partnership;** Encouraging smaller manufacturers to collaborate with institutions, academia and research organizations will grant them access, to datasets and expertise fostering better development practices.
4. **Tailored Ethical Guidelines;** Developing and implementing guidelines that are specifically tailored for manufacturers will serve as a comprehensive framework guiding their responsible development and deployment processes.

3.5.4. Recommendations and Guidelines for Small Vendors Developing Localized AI/ML Models

Support resources and guidelines are crucial for addressing ethical concerns related to localized development by smaller manufacturers while fostering innovation and diversity in the field. Below are examples of such resources and guidelines:

3.5.4.1. Support Resources

1. Data Sharing Platforms

- Description: Establish platforms where healthcare institutions, researchers, and manufacturers can share and access diverse and representative datasets.

- Purpose: To check that smaller manufacturers have access to the data needed for developing unbiased and generalizable algorithms.

2. Funding and Grants (not just Broad Agency Announcements)

- Description: Provide financial support through grants, subsidies, or low-interest loans specifically aimed at supporting ethical AI/ML development in healthcare.
- Purpose: To alleviate financial constraints that might hinder the ethical development and validation of AI/ML algorithms by smaller manufacturers.

3. Technical Assistance

- Description: Offer technical support and assistance to smaller manufacturers in the development, validation, and deployment of AI/ML algorithms.
- Purpose: To help smaller manufacturers navigate technical challenges and implement best practices in algorithm development and validation.

4. Educational Resources

- Description: Develop and disseminate educational materials, workshops, and training programs on ethical AI/ML development in healthcare.
- Purpose: To educate and inform smaller manufacturers about the ethical considerations and best practices in AI/ML development.

3.5.4.2. Ethical Guidelines

1. Ethical Principles for AI/ML Development

- Description: Establish a set of ethical principles that guide the development and deployment of AI/ML algorithms in healthcare, focusing on fairness, transparency, accountability, and non-discrimination.
- Purpose: To provide a foundational ethical framework for smaller manufacturers to adhere to during the development process.

2. Best Practices for Data Management

- Description: Outline best practices for data collection, management, and usage, emphasizing the importance of data diversity, representativeness, privacy, and security.
- Purpose: To guide smaller manufacturers in managing data ethically and responsibly.

3. Algorithm Validation and Testing Guidelines

- Description: Develop guidelines for the validation and testing of AI/ML algorithms, including recommendations for external validation and performance assessment.
- Purpose: To check that algorithms developed by smaller manufacturers are rigorously tested and validated for safety and efficacy.

4. Transparency and Accountability Standards

- Description: Set standards for transparency and accountability in AI/ML development, including requirements for documentation, explanation, and justification of algorithmic decisions.

- Purpose: To promote transparency and accountability among smaller manufacturers in the development and deployment of AI/ML algorithms.

5. Community Engagement Recommendations

- Description: Provide recommendations for engaging with patient communities, healthcare providers, and other stakeholders during the development process.
- Purpose: To foster collaboration and input from the broader community to ensure that AI/ML algorithms address the needs and concerns of all stakeholders.

Part III. Development of IKM

Table of Contents

4. Framework Considerations for Knowledge Management (KM)	37
4.1. Framework Considerations Purpose	37
4.2. Introduction to KM Processes	37
4.3. Current State Analysis	37
4.4. Methodology	37
4.4.1. Stakeholder Interviews	37
4.4.2. Requirements, Architecture, and Documentation Review	37
4.4.3. Code Review	38
4.5. Findings	38
4.5.1. Requirements Review	38
4.5.2. Architecture and Documentation Review	38
4.5.3. Code Review	38
4.5.4. Duplication	39
4.5.5. Layers and Coupling	40
4.6. Technology Choice Framework	40
4.7. Code Maturity	40
4.7.1. Documentation and Style Guidelines	40
4.7.2. Code Reviews	40
4.7.3. Immutability	43
4.7.4. How to Write Code Review Comments	45
4.7.5. Testing and the Cost of Defects	46
4.8. Automation	52
4.8.1. Continuous Integration (CI)	52
4.8.2. Continuous Deployment (CD)	54
4.8.3. Packaging and Distribution	54
4.8.4. Expected Changes to CI/CD	55
4.9. Scalability	55
4.10. License	55
4.11. Security	55
4.12. Privacy Needs	56

4. Framework Considerations for Knowledge Management (KM)

4.1. Framework Considerations Purpose

This document will serve to identify a framework of considerations for comparing KM technologies. Incorporated within is analysis of the architecture and codebase for any known scalability, licensing, and privacy needs, as well as how to manage these non-functional requirements continuously as the product develops. In doing this, we will make suggestions for software solutions to cover code maturity, automation, and security.

4.2. Introduction to KM Processes

As we progress along our journey to improve the open-source software base for a reference implementation for IKM capabilities, it is important to take note of what is currently being done and what we will focus on as we develop this work further. As part of this observation, we are defining the processes that needs to be executed to achieve the goals of this work, such as becoming a well-supported and contributed open-source product that encourages adoption by the larger community.

4.3. Current State Analysis

As part of the work on the KM tooling, we first need to document the current state of the project so that we can define a future state and develop a plan to address any concerns. To do this, we have completed code reviews and stakeholder interviews to ensure that we understand what work is outstanding.

4.4. Methodology

The following section describes how we approached reviewing the current assets as it relates to development of the Knowledge Management suite of tooling.

4.4.1. Stakeholder Interviews

Stakeholder interviews were conducted on identified individuals that had either prior experience with the existing KM code base (e.g., Komet, ISAAC, Tinkar-core) or the Informatics Knowledge Architecture (e.g., a description-logic based integration of SNOMED CT®, LOINC®, and RxNorm [Solor]). A basic semi-structured thematic analysis was performed to identify qualitative themes from the interview.

4.4.2. Requirements, Architecture, and Documentation Review

Analysis included reviewing the website documentation, READMEs, any ancillary postscript/PowerPoint/etc. documentation that is available to determine how a customer would approach using the software and how an open-sourced developer would approach to contributing to the codebase.

Analysis included reviewing any documentation that discussed what the project must do and must continue to do. This could be contained in provided requirements lists or architecture diagrams. Architecture

diagrams include (but are not limited to) logical and physical architecture, component diagrams, sequence diagrams, interaction diagrams, and data flow diagrams. Analysis will also include validating how well the codebase executes functional and non-functional requirements, as so far as they are described in the requirements.

4.4.3. Code Review

Analysis of code included doing a full code review, as is described in further sections. This code will determine how well documented, object-oriented (OO) (when applicable), layered, and modular the code is. Analysis will also include validating the frequency in which automated unit and integration testing is being performed on the codebase.

4.5. Findings

After interviews with the stakeholders and the review of the codebase, we determined that the following parts of the codebase must be addressed to make it useable by this team and the greater community.

4.5.1. Requirements Review

There is a general lack of both functional and non-functional requirements. A lot of promotional material regarding the design of Solor and its Knowledge Architecture exist, from which some functional requirements could be inferred, but this isn't sufficient to develop highly reliable KM capabilities. There are some existing Jira and Confluence sites from Logica (formerly HSPC) that describe some groupings of capabilities and functionality (e.g., Personas, SNOMED CT® export, Simple Search) but not enough detail is present to be considered requirements user stories can be based off.

Additionally, there is a lack of Key Performance Indicators and Service Level Agreements as they pertain to the larger idea of a reference implementation for IKM capabilities that will be host to and a conduit of resolving extremely complex knowledge standards-based issues within the health IT landscape.

4.5.2. Architecture and Documentation Review

Identified through various stakeholder interviews were Unified Modeling Language (UML) diagrams that detailed some aspects of Tinkar data structure and overall class hierarchy. These were found within HL7 informational ballot for Tinkar and some HL7 Vocabulary working group presentations. There were also some identified attempts at using JetBrains IntelliJ IDEA IDE to auto-generate UML class diagrams of core Tinkar concepts.

As mentioned previously, some pre-existing Jira and Confluence websites (Logica) were identified and found to contain some outdated information regarding the Knowledge Architecture (PASTF diagram).

4.5.3. Code Review

In general, there is no formal external documentation describing code functionality – either at the class or method level. Some Java Classes were identified to have properly formatted JavaDoc annotations, but this wasn't consistently applied across the entire code base. Below the unit test code was examined using a Java-based code coverage tool (JaCoCo)

Figure 4.1. Unit Test Code Coverage Report for Komet Using JaCoCo














Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 classification	<div><div></div></div>	49%	<div><div></div></div>	38%	8,391	14,634	15,151	30,027	4,369	8,978	405	1,361
 framework	<div><div></div></div>	0%	<div><div></div></div>	0%	3,414	3,414	9,290	9,290	2,207	2,207	236	236
 navigator	<div><div></div></div>	0%	<div><div></div></div>	0%	479	479	1,248	1,248	261	261	23	23
 details	<div><div></div></div>	0%	<div><div></div></div>	0%	225	225	823	823	162	162	17	17
 preferences	<div><div></div></div>	0%	<div><div></div></div>	0%	388	388	910	910	266	266	17	17
 list	<div><div></div></div>	0%	<div><div></div></div>	0%	161	161	424	424	111	111	10	10
 application	<div><div></div></div>	0%	<div><div></div></div>	0%	71	71	325	325	55	55	8	8
 progress	<div><div></div></div>	0%	<div><div></div></div>	0%	84	84	252	252	63	63	9	9
 executor	<div><div></div></div>	0%	<div><div></div></div>	0%	68	68	139	139	51	51	8	8
 search	<div><div></div></div>	0%	<div><div></div></div>	0%	34	34	69	69	30	30	3	3
 komet-terms	<div><div></div></div>	0%	<div><div></div></div>	n/a	2	2	45	45	2	2	1	1
 builder	<div><div></div></div>	0%	<div><div></div></div>	0%	26	26	46	46	20	20	2	2
 artifact	<div><div></div></div>	0%	<div><div></div></div>	n/a	36	36	38	38	36	36	4	4
Total	128,140 of 192,582	33%	9,906 of 13,995	29%	13,379	19,622	28,760	43,636	7,633	12,242	743	1,699

Figure 4.2. Unit Test Code Coverage Report for Tinkar using JaCoCo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes	
entity	<div><div></div></div>	0%	<div><div></div></div>	0%	2,082	2,082	4,205	4,205	1,339	1,339	116	116	
coordinate	<div><div></div></div>	0%	<div><div></div></div>	0%	1,586	1,586	2,784	2,784	1,128	1,128	110	110	
common	<div><div></div></div>	8%	<div><div></div></div>	7%	1,277	1,364	2,239	2,449	708	774	91	108	
terms	<div><div></div></div>	0%	<div><div></div></div>	0%	105	105	1,854	1,854	65	65	12	12	
collection	<div><div></div></div>	21%	<div><div></div></div>	21%	990	1,189	2,116	2,758	423	541	51	62	
dto	<div><div></div></div>	36%	<div><div></div></div>	33%	536	770	804	1,381	394	573	40	64	
data-spinedarray-provider	<div><div></div></div>	0%	<div><div></div></div>	0%	205	205	620	620	114	114	15	15	
data-mystore-provider	<div><div></div></div>	0%	<div><div></div></div>	0%	91	91	208	208	59	59	6	6	
entity-provider	<div><div></div></div>	0%	<div><div></div></div>	0%	87	87	190	190	59	59	6	6	
data-websocket-client-provider	<div><div></div></div>	0%	<div><div></div></div>	0%	62	62	148	148	54	54	2	2	
data-eohermal-provider	<div><div></div></div>	0%	<div><div></div></div>	0%	64	64	134	134	41	41	2	2	
executor-provider	<div><div></div></div>	0%	<div><div></div></div>	0%	57	57	117	117	40	40	6	6	
search-provider	<div><div></div></div>	0%	<div><div></div></div>	0%	21	21	97	97	13	13	2	2	
component	<div><div></div></div>	72%	<div><div></div></div>	87%	21	55	22	92	17	21	8	9	
data-websocket-server-provider	<div><div></div></div>	0%	<div><div></div></div>	n/a	7	7	23	23	7	7	1	1	
integration	<div><div></div></div>	0%	<div><div></div></div>	n/a	3	3	12	12	3	3	2	2	
Total		75,103 of 81,390	7%	5,101 of 5,618	9%	7,194	7,748	15,573	17,072	4,464	4,831	470	523

There is a general lack of unit tests across the entire code base. As shown in [Figure 4.1, “Unit Test Code Coverage Report for Komet Using JaCoCo”](#) and [Figure 4.2, “Unit Test Code Coverage Report for Tinkar using JaCoCo”](#) above, JaCoCo only identified about 25.8% of functional code as adequately covered by unit tests. In contrast, a Maven module within the Tinkar-core codebase called “integration” was identified as containing several large integration tests. These integration tests currently use Junit5 (historically TestNG) to build test suites to validate that the three main Tinkar providers Ephemeral, multi-version store (MVStore), and Spined Array properly load a preset test sample of Tinkar data. The selection of and adequate testing suite is important because the codebase utilizes inversion of controls (or dependency injection) to start appropriate Tinkar-core “services” which requires the testing suite to have proper setup and tear annotations.

Overall, the codebase had low code maturity. This is attributed to a lack of continuous integration and continuous delivery infrastructure. It was identified that historically a previous version of the codebase utilized the open-source version of Travis-CI. Currently no CI/CD tooling is being used on the codebase.

4.5.4. Duplication

There was little to no duplication identified within the codebase.

4.5.5. Layers and Coupling

Overall, the codebase has loose coupling and high cohesion. This is supported by a multi-module Maven architecture – which helps to isolate complexity by grouping similar classes into reasonably sized modules and then enforcing reasonable degree of interdependence between those modules.

4.6. Technology Choice Framework

As the teams create solution architectures for the products, we will inevitably need to determine what technologies are needed. When evaluating technology on the project, we will first define the use case in the architecture. Then, the team will create an Analysis of Alternatives to review with stakeholders. This analysis of alternatives must include resource and license costs as well as its fit with the system requirements. For example, if it needs to be an extremely fast system and must use a key-value read pattern, it will be defined empirically, when possible, how fast it is and how well it fits that pattern.

4.7. Code Maturity

As the codebase improves, our analysis showcased a general need to improve Code Maturity. Code Maturity covers many areas, but primarily will be measured by how modular and maintainable the codebase is and how likely the code is to continuously meet functional and non-functional requirements upon release.

To achieve a maintainable product, we need to make sure the documentation is thorough and understandable, allowing for new developers to analyze the codebase and contribute fixes to defects. Our processes must provide procedures for contributing code to the codebase, encouraging feedback, and providing avenues for support. The procedures must allow for traceability and reproducible actions, enforcing consistency the output products.

To guarantee that the code meets functional and non-functional requirements, one of the most important steps is automated and manual testing. Testing provides the ability to validate that functional and non-functional requirements are being met and regression test when the code changes. There are many types of tests, and the ones that the teams will focus on are:

- Unit Tests
- Integration Tests
- End-to-End Functional tests
- API Tests
- Performance Tests

4.7.1. Documentation and Style Guidelines

As we move forward, we will incorporate documentation and style guideline checks to encourage teams to adhere to standard practices for improved collaboration and understandability of code.

4.7.2. Code Reviews

Code reviews are a way to ensure that the features being delivered match the request and the architecture, to improve code quality and save time. This is because code reviews:

- make for better estimates.

- enable time off.
- mentor newer engineers.

We suggest doing code reviews inline as well as at a regularly scheduled interval. For example, if you are using GitHub/GitLab, you should perform inline reviews on the code pull/merge request, preferably keeping traceability via approvals. This can be done by any team member but ideally those with the next most knowledge around a subject. Then, on a weekly cadence, we suggest doing a review of stories that is or has been closed so that everyone on the team has exposure to the changes.

In doing a code review, you should make sure that:

- terms and variables semantically match the domain.
- code is well-designed.
- functionality is good for the users of the code.
- any UI changes are sensible and look good.
- any parallel programming is done safely.
- code isn't more complex than it needs to be.
- the developer isn't implementing things they might need in the future but don't know they need now.
- code has appropriate unit tests.
- tests are well-designed.
- the developer used clear names for everything.
- comments are clear and useful and mostly explain *why* instead of *what*.
- code is appropriately documented (generally in g3doc).
- code conforms to our style guides.
- developers haven't (inadvertently or not) subverted code quality checks.

Make sure to review **every line** of code you've been asked to review, look at the **context**, make sure you're **improving code health**, and compliment developers on **good things** that they do.

Some of these require some nuance, so we'll go into further detail below.

4.7.2.1. Functionality

When executing a code review on a story, the team member should review the story and acceptance criteria and make sure that the functionality described matches the code that has been delivered.

4.7.2.2. Code Design

It is important that code has a simple, cohesive and intuitive design. Is the code more complex than it should be? Are functions/classes too complex? "Too complex" usually means "can't be understood quickly by code readers". It can also mean "developers are likely to introduce bugs when they try to call or modify this code".

A particular type of complexity is over-engineering, where developers have made the code more generic than it needs to be or added functionality that isn't presently needed by the system. Reviewers should be especially vigilant about over-engineering. Encourage developers to solve the problem at hand, not the problem that the developer speculates might need to be solved in the future. The future problem should be solved once it arrives, allowing visibility into its actual shape and requirements in the physical universe.

4.7.2.3. Duplication

It is important in reviewing complexity to review duplication. Duplication means that you will have to update code in multiple places and fix defects in multiple places. "Don't Repeat Yourself" (DRY), is a way to encourage code reuse and increase code quality by focusing improvements on a single area.

While you can improve code sharing, it is important that you do not tightly couple your delivery. DRY was originally intended for systems, not for code duplication, though it tends to fit well. In general, it is more acceptable to have code duplicated rather than tightly coupling deliveries – which ultimately produces a monolithic application that is only delivered as a single unit.

4.7.2.4. Ensuring Code Follows the Appropriate Principles

When reviewing OO code, it is important to ensure that it meets the principles below:

- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion (DI)

These are known as SOLID Principles and, as Robert C Martin described, OO programs should obey these principles.

4.7.2.5. Layers and Coupling

To ensure that code remains reusable and allows teams to pivot to respond to business needs, it is important to create programs that are loosely coupled and thus modular. Modularity allows the codebase to be used in different ways and allows teams to pivot the direction of the project more easily.

This is often inferred as part of ISP and DI, but it is important that this is part of the solution architecture review process in code reviews.

Some simple rules of thumb include:

- Data and its representation should be in a layer all by itself, separate from the business logic.
- If providing a UI, it is important to abstract the UI away from the business logic.
- Generally, the team suggested using the Composition Pattern over inheritance unless the substitution principle of LSP is met.

4.7.2.6. Adherence to Patterns

Design Patterns allow developers to solve common problems and engineers to speak a common language. There are sets of patterns for Construction Objects, Structuring Data, and Behavioral Patterns. As this is beyond the scope of this document, it is important for developers to seek these out.

4.7.3. Immutability

Immutable classes are those that do not change after creation. Immutability, though it often can cause extra object creation, allows for several benefits. The key is knowing when you should be using immutability. The following guide should be used to help determine when immutability should be used.

“Classes should be immutable unless there's a very good reason to make them mutable... If a class cannot be made immutable, limit its mutability as much as possible.” – Joshua Bloch, from *Effective Java*

Benefits of Immutability

- Thread-safe, preventing synchronization issues
- Good Map keys and Set elements, since these typically do not change once created
- Easier to write, use, and reason about the code (class invariant is established once and then unchanged)
- Effortless to parallelize your program as there are no conflicts among objects
- The internal state of your program will be consistent even if you have exceptions
- References to immutable objects can be cached given their static nature

Drawbacks of Immutability

- Due to creating more objects, there was a copy cost and often a cleanup cost via garbage collection, which could impact the performance of a single Java virtual machine. It is generally suggested that the benefits provided by additional multithreading often outweigh the performance costs.

Items that should be immutable:

- Data Transfer Objects (DTOs)
- Domain/Data Object Model (DOM) Objects
- Services (contain behavior and business logic)

4.7.3.1. Multithreading

Multithreading is a complex topic and requires the use of complex math to prove that most multithreading really works with locking. It is suggested that, if you are reviewing code using major multithreading, you utilize established libraries with millions of eyes on them as this will be more effective than home-grown locking mechanisms. In reviews, just make sure that this is the case. For a book on the topic, please refer to the book *Java Concurrency in Practice*, the bible of Java concurrency. It explains just how complex this is. The author of this book was integral in creating the Akka framework, which is heavily used in Java, for example.

4.7.3.2. Documentation

Since the biggest component of complexity is how easily a new developer can pick a piece of code up, it should be obvious that comments and repository documentation are integral to review. The documentation should be reviewed as part of each code submission to make sure that it matches the updates. The following subsections will cover in detail some of the portions of the documentation.

The README file is the file at the top level of a repository, usually written in markdown or text format. It should be the entry point to all documentation and contain anything that a developer should know to

get started using a project. If this project is a library, it should contain ample examples of how to use the product.

Documentation should also include everything required for understanding the architecture of the codebase. By coupling the documentation with the code, it should be obvious when the documentation is out-of-date and needs updating. This documentation can then be distributed separately, if needed. This should include anything that makes it clear to developers and end-users how to develop and deploy the product.

Often, with things like libraries, you will have more documentation than fits in one folder or you will have referenced images. It is suggested that you create a "docs" folder at the top of your repository and have additional markdown and images that can be referred to there.

4.7.3.3. Comments describing Code

Code surrounding classes and methods should be fully describe the functionality of the single operation object or method.

- **Package/Namespace Comments** - Some programming languages let you document groups of classes. This should be a high-level description and relatively brief. If you must go into detail, you may need to break your packages/namespaces up further. An example is shown in [Figure 4.3, “Example of Package/Namespace Comments”](#).

Figure 4.3. Example of Package/Namespace Comments

```
/**
 * Group of classes that is used for parsing
 * json from files.
 */
package a.b.c;
```

- **Class Comments** - Used for describing the single purpose for a class. There are many parameters in Javadocs that should be used in a Java example, but most relating to version, since timeline and author should be avoided. Those should be left to version control. Any function or class parameters are required. Deprecation is a good notice, too, as it will indicate when a class may soon be retired and replaced with another class. An example is shown below.

Figure 4.4. Example of Class Comments

```
/**
 * The HelloWorld class holds all methods related
 * to execution of a program
 * that simply displays "Hello World!" to the
 * standard output.
 *
 * @param <T> - example of a template parameter in documentation
 * @deprecated use {@link HelloWorldNew} instead
 */
public class HelloWorldOld<T> {

    //...

}
```

- **Method/Function Comments** - Comments directly related to calling of a function or method. These often contain a usage description, the description of the input and output values, and any information about deprecation or templating. An example of this is shown below.

Figure 4.5. Example of Method/Function Comments

```
/** ...
 */
public class HelloWorld {
    /**
     * An example method that does nothing but
     * showcase usage.
     *
     * @param firstString the parameter that will be
     *     printed first and saved for later
     * @param secondValue the parameter that will be
     *     printed second
     * @returns the number of things printed
     *
     * @deprecated use {@link HelloWorldNew#example(String, int)} instead
     */
    public int example(final String firstString, final int secondValue){
        int printedValues = 0;
        //...
        return printedValues;
    }
    //...
}
```

- **Block Comments** - Comments that describe portions of code that can be used to selectively execute certain blocks. Block comments should be avoided if possible. If a section of code is so complex that it needs a block comment, it probably needs to be its own method/function/class/etc.
- **Inline Comments** - Step-by-step details about what you are doing. These can be used judiciously, provided they make sense. An example of an inline comment is show in [Figure 4.6, “Example of Inline Comment”](#) below.

Figure 4.6. Example of Inline Comment

```
// increment variable
variable++;
```

4.7.3.4. Commented Out Code

There should be no commented-out code and no functionality that was not in the requirements. This is the job of Version Control, and you should always be removing it, not commenting it out.

"Always implement things when you actually need them, never when you just foresee that you need them."
– Ron Jeffries (XP Co-Founder)

4.7.4. How to Write Code Review Comments

When writing code review comments:

- be kind.
- explain your reasoning.
- balance giving explicit directions with just pointing out problems and letting the developer decide.
- encourage developers to simplify code or add code comments instead of just explaining the complexity to you.

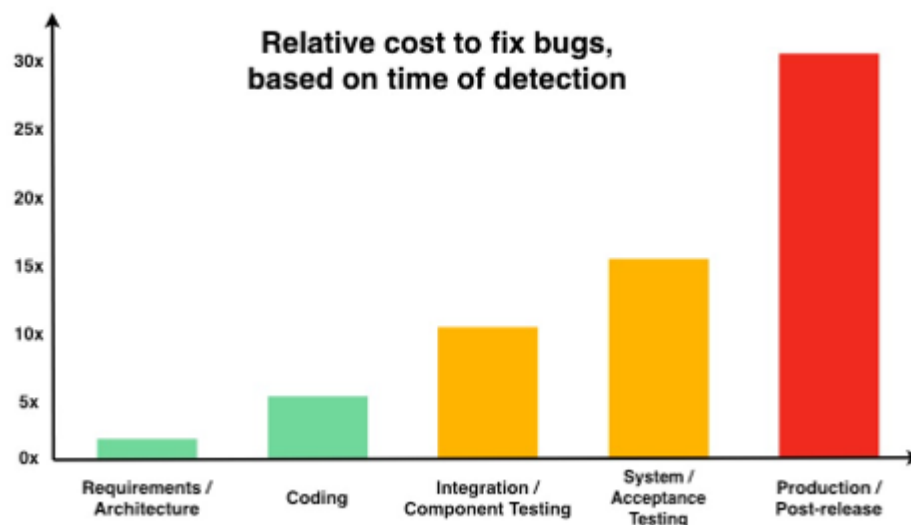
4.7.5. Testing and the Cost of Defects

The cost of defects can be measured by the impact of the defects and when we find them. The earlier the defect is found, the cheaper it is to fix. Kent Beck, a leader in the field of software testing said, via his book *Extreme Programming Explained*

“Most defects end up costing more than it would have cost to prevent them. Defects are expensive when they occur, both the direct costs of fixing the defects and the indirect costs because of damaged relationships, lost business, and lost development time.” – Kent Beck, from *Extreme Programming Explained*

The following graph, courtesy of **the National Institute of Standards and Technology**, helps in visualizing how the effort in detecting and fixing defects increases as the software moves through the five broad phases of software development.

Figure 4.7. Relative Cost to Fix Bugs, Based on Time of Detection



For example, in the case of a data storage error:

- If the error is found in the requirements gathering, then it is relatively cheap to fix. The correction can be made to the requirements and then code can be re-issued. Any active architecture designs can be adjusted.
- If the error is found in development (defect), then the developer must work with the requirements team to figure out how the requirements need to be change. The defect can be corrected by refactoring and or rewriting the problematic code to accommodate new requirement changes.
- If the error is found in production, then the requirements team needs to be consulted with to adjust requirements and architecture, development needs to determine refactoring, a cost to schedule must be

determined. Additionally, if this service has done data processing, then that needs to be designed and developed. Both the application and the correction to the data processing need to go through testing now. Backups need to be made of the data so that if there is an issue, the original data can be recovered, as this now has the business risk of impacting production users.

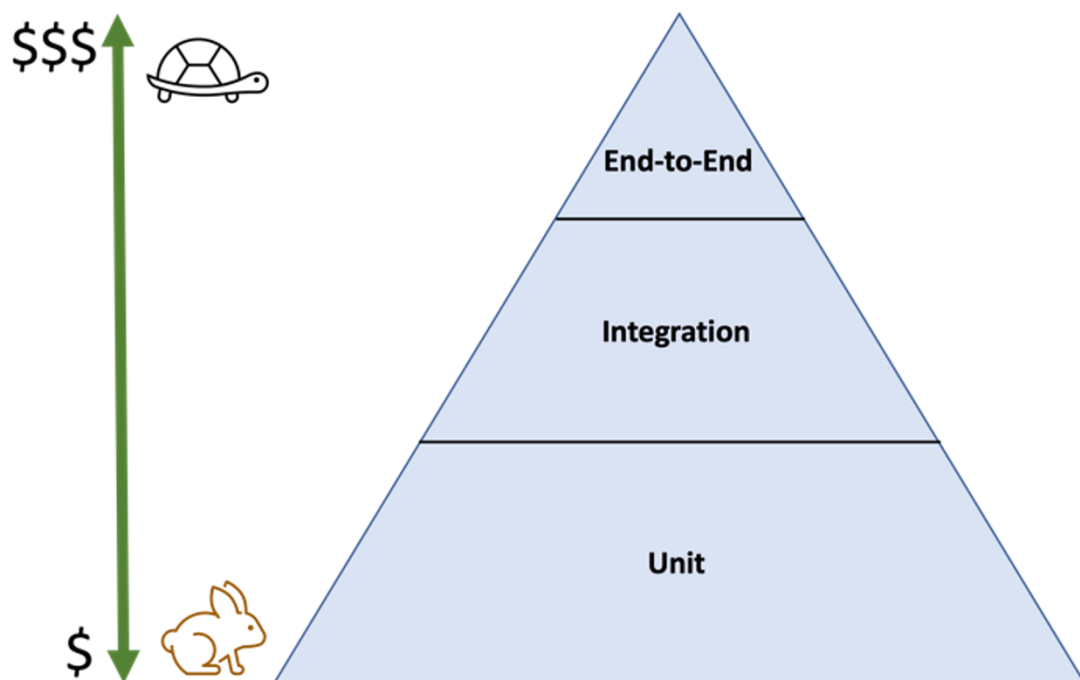
There will always be defects in production, but the key is to reduce them to the ones that can't be predicted. The mantra of any developer should be to *detect early and often*.

4.7.5.1. Right Size the Testing

Given the above information, it is common to want to completely switch gears and test until every possible bug is caught with extremely large end-to-end suites. Unfortunately, this is also a huge cost.

As Kent Dodd and Martin Fowler suggest, and as shown in [Figure 4.8, “Monetary and Time Costs of Testing Types”](#), testing costs are significantly higher and slower to develop and execute at more thorough levels. Therefore, we should spend more time at unit and integration tests to iterate quickly and resolve most issues, using end-to-end tests sparingly, as they are costly.

Figure 4.8. Monetary and Time Costs of Testing Types



When doing a story tasking as part of Iteration Planning, teams should create tasking for integration and end-to-end tests that consider the amount of time that you need to thoroughly test the acceptance criteria. This should be the minimal amount required to achieve proving the criteria have been met.

4.7.5.2. Test Driven Development (TDD) Standards

Pipelines not only encourage Test Driven Development (TDD) but will require it by default out of the box. Pipelines will have a requirement of 85% code coverage which comprises a combination of line coverage, package coverage, and branching logic.

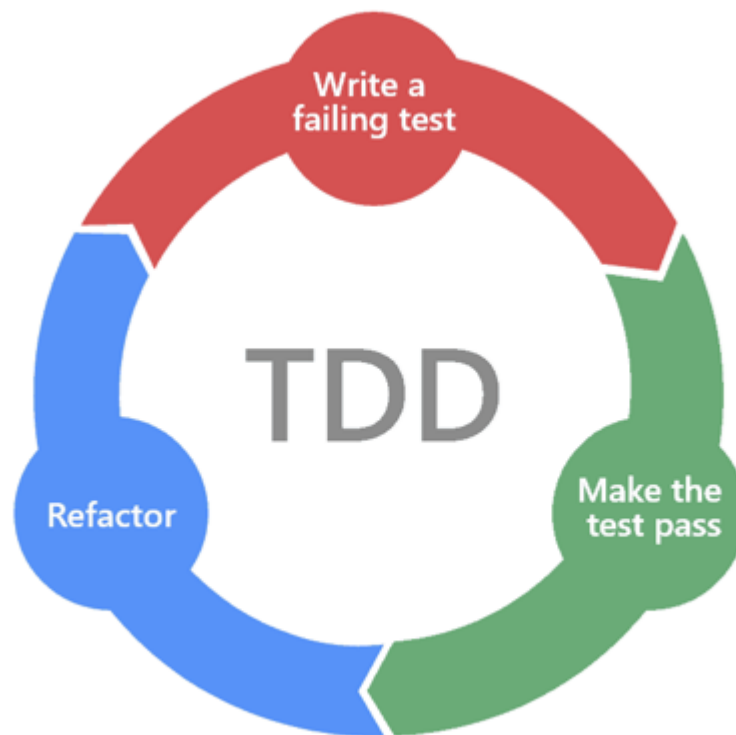
All builds will index reports generated back to SonarQube which will ultimately enforce the coverage requirements, so long as the SonarQube stage is in the build.

4.7.5.3. Test Driven Development

TDD is an idea that was introduced by Kent Beck. This is a principle of defining requirements before writing code. How this translates to modern development is that it is expected that this is the first thing that you do when you pick up a story. This is a set of tests that should match the acceptance criteria and showcase to the product owner that the story is complete.

TDD is a simple three-phase cycle of writing something that is failing, making a test pass, then refactoring, and repeating. This process is depicted in [Figure 4.9, "TDD Cycle Diagram"](#).

Figure 4.9. TDD Cycle Diagram



4.7.5.3.1. Writing Failed Tests (Red)

This should consist of the following tests, every time:

- Functional Tests - At least one functional test should be performed, but likely more than one will be needed.
- Positive (also known as "Happy Path") Testing - Positive Testing proves that the code fulfills its intended duties.
- Negative Testing - One or more scenarios that prove something that your code should not accept or handle gracefully. This often includes input validation problems or SQL Injection problems. It also includes things that should not be allowed from a business perspective. In a shopping cart application, this would be as if the user abandoned a transaction, if the user tries to purchase the same thing on two laptops when only one is allowed, or if there is a failed payment. All of these should be recoverable.

Once these tests are created, the development team can start working on the stories using the Red-Green-Refactor Loop. This means that all these tests fail initially - the software doesn't currently support these actions.

4.7.5.3.2. Writing the Solutions (Green)

Developers start by picking a test and working on it. You should be doing the minimum amount to make the test pass. This is confusing to some people, as they want to create all functionality upfront. The problem with this is that you may not be able to finish your story within the sprint time frame, and as developers, we want to ensure that test requirements are met before pivoting.

Example:

The requirement is that you create a myArrayList application. [Figure 4.10, “PseudocodeforCreatingMyArrayList”](#) below contains some pseudocode for that test.

Figure 4.10. PseudocodeforCreatingMyArrayList

```
given
  Create MyStringArrayList
then
  check size == 0
when
  add value to MyStringArrayList
then
  give the size of 1
```

Then your code should be the easiest way to achieve this, which is shown in [Figure 4.11, “Simplest Code for Creating the MyArrayList Application”](#) below.

Figure 4.11. Simplest Code for Creating the MyArrayList Application

```
public class MyStringArrayList {
    int i = 0;

    /**
     * Meets the base requirements of showing 0 after the first call and 1 after the second
     * @return the number of times it has been called minus 1
     */
    public int getSize() {
        i++;
        return (i - 1);
    }

    /**
     * There are no requirements around the adding of values yet. Don't add functionality until you get testable
     * requirements.
     */
    public void addValue(String s) {
        //do nothing
    }
}
```

By doing it this way, it is very clear where logic should be adjusted and separated out, as different methods will become immediately apparent where they need modularity.

4.7.5.3.3. Making the Code Reusable (Blue)

Realistically, we know that's not the long-term solution to this problem. This is where you spend some time refactoring in preparation for the next test(s). You can move your code around, reorganizing your tests in methods and classes that make more logical sense now that you have gotten a passing test.

Questions you should ask during a refactor are:

- Can I make my test suite more expressive?
- Does my test suite provide reliable feedback?
- Are my tests isolated?
- Can I reduce duplication in my test suite or implementation code?
- Can I make my implementation code more descriptive?
- Can I implement something more efficiently?

4.7.5.3.4. Code Coverage Philosophy

Testing for the purpose of code coverage is malpractice, developers would have circumvented the intentions of TDD if the goal of testing is to show green lines and high percentages in code coverage reports. If development teams are retroactively writing test cases, code quality is harder to enforce as testing can easily be hacked to show high coverage metrics. **Quality and high code coverage is a byproduct of proper testing practices like TDD.**

A typical guideline for code coverage:

- Function/Method coverage → 100%
- Package/Class coverage → 80% to 100%
- Line Coverage → 80% to 100%
- Branch coverage → 80% to 100%
- Overall → 85% or better

Arguably code coverage is not the best metric, as it can often represent pieces of the code that can never be executed. However, it is still the best tool to validate that the code has been tested thoroughly enough. It will be used as a gate for guidance, and we can tweak and ignore code blocks if needed.

4.7.5.3.5. Behavior Driven Development (BDD)

Behavior Driven Development (BDD) is a spin on TDD. Where TDD focuses on testing a specific piece of functionality, BDD is focused less on a piece of functionality, and more on a user action. BDD should generally be used in conjunction with TDD, as their goals are slightly different. UI, for example, should primarily utilize BDD over TDD to test requirements.

A collaboration between functional and development teams to create feature development solution expectations driven by test case definitions prior to development. The goal of BDD is to ensure that functional and business teams have input and visibility into the software development process, and in turn development teams will have clearer understanding on what the expected solutions should accomplish. BDD is great when there is a need for functional teams to be involved with designing and testing features. BDD

requires test cases to be written in Gherkin syntax which is then translated to code. Gherkin (format → Given, When, Then) is readable to functional teams, hence providing visibility. An example of BDD is shown below in [Figure 4.12, “Example of Behavior Driven Development”](#).

Figure 4.12. Example of Behavior Driven Development

```
Scenario: Breaker guesses a word
Given
  the Maker has chosen a word
When
  the Breaker makes a guess
Then
  the Maker is asked to score
```

BDD can be practiced using most testing frameworks. **Cucumber**, however, is a common tool used in industry as it provides, amongst many features, .dsl files which are maintained by the functional team. Note: Use the framework/methodology that works for your team. Given the descriptions in this article, teams should be able to discern what will work for their projects.

4.7.5.3.6. UI Testing

UI testing is to ensure that users have a consistent visual user experience across a variety of platforms and that the user interaction is consistent with the function requirements. This can apply to simple tasks, such as validating that the logo is in the correct place on the computer and the iPhone.

UI testing should:

- ensure the UI appearance and interaction satisfy the functional and non-functional requirements.
- detect changes in the UI both across devices and delivery platforms and between code changes.
- provide confidence to designers and developers the user experience is consistent.
- support fast code evolution and refactoring while reducing the risk of regressions.

The scope of UI testing should be strategic. UI tests can take a significant amount of time to both implement and run, and it's challenging to test every type of user interaction in a production application due to the large number of possible interactions.

Designing the UI tests around the functional tests makes sense. For example, given an input form, a UI test would ensure that the visual representation is consistent across devices, is accessible and interactable, and is consistent across code changes.

UI Tests will catch 'runtime' bugs that unit and functional tests won't. For example, if the submit button for an input form is rendered but not clickable due to a positioning bug in the UI, then this could be considered a runtime bug that would not have been caught by unit or functional tests.

4.7.5.3.7. Accessibility Testing

Designers should focus on promoting UIs and designs that are accessible for all users. Accessibility considerations should be reviewed for all wireframes before considered for development. More than general accessibility, there is often strict accessibility standards that are part of requirements. For example, many government sites encourage accessibility by requiring 508, Americans with Disabilities Act (ADA) or Web Content Accessibility Guidelines (WCAG) compliance. This should be tested for in an automated way.

Once a user interface is ready for development, the developer should be conscientious of accessibility when constructing the user interface. As part of the automated functional tests, after each spec, an accessibility test should be run automatically to check for any possible violations. Once the automated test is completed, a report should be created and attached to every build in the pipeline. This report should detail each test case as well as any possible accessibility violations, the location of the violation, and steps to remediate the issue.

4.7.5.3.8. Test Suites

These testing suites will be used in the pipeline access code coverage in the pipeline. In the seeds provided, these can also work to execute these tests locally. The following showcases the results of an Analysis of Alternatives based on the language and framework being used.

These include (but are not limited to):

- JUnit 5 and JaCoCo (Java)
- Jest (JavaScript and React)
- Karma and Jasmine (Angular)

Our team suggests leveraging Cypress for runtime Web UI testing and Electron Applications to test cross-browser compatibility.

4.8. Automation

CI and CD Processes, [Figure 4.13, “Framework for Continuous Integration”](#) and [Figure 4.14, “Framework for Continuous Deployment”](#), respectively, allow teams to be productive from day one, even releasing as early as their first day. Initial tooling will allow deployments of software and restrict access to the tooling that is needed for licensing purposes. As the necessary materials for a fully supported Open-Sourced project are created, we will re-evaluate parts of this tooling to allow for teams to pivot some or all the DevOps processing to SaaS (Software as a Service), based on overarching pricing and supportability, making sure that this allows us to create a culture of promoting code submissions and foster access to this codebase to improve it over time.

4.8.1. Continuous Integration (CI)

Figure 4.13. Framework for Continuous Integration



The current Software Factory installation allow teams to do the following:

- **Plan** - Processes have been developed to help teams work together and produce code quickly. Integrations have been created in API driven systems to guarantee team’s traceability from story creation to code deployment and everything in between, including:
 - Integration with JIRA Issue Tracking
 - Automatic tie of JIRA stories to feature Branches

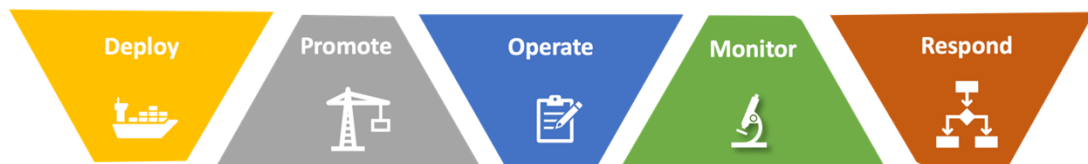
- **Develop** - Using Seed projects, teams can start coding in most popular languages from day one with batteries included for best practices and security, targeted for deployment with containers. By utilizing the development standards created here, it becomes easy to create code and release your first version in minutes.
 - Consistent Project Setup - Titan Command Line Interface (CLI) will create and enforce defined projects
 - GitOps - All processes use code to define state, giving full traceability from Requirements through to delivery.
 - Git Triggers for enforcing Traceability
 - Enable Secure Development from an internal Repository
 - Version Control Branching standards, define Team Code Reviews, and enforce Squash Merge settings
 - Build Tool and integrated development environment (IDE) integration - Comes with instructions on how to adjust Build tooling to use internal repositories for higher security
- **Build** - By creating installations of powerful DevSecOps tooling that is preconfigured for use, our team creates a process that is defined for compiling and packaging that allows you to build agnostic to your deployment environment, making it very easy to shift to another cloud or Kubernetes deployment.
 - Jenkins preconfigured to build utilizing Kubernetes Agents
 - Pre-created build containers for use with Kubernetes Build Agents
 - Team-based Project scanning to scan a Team's projects and automatically create and execute multi-branch jobs
- **Test** - The tooling has appropriate plugins and integration installed, paired with seeds, which make it simple to locate and execute unit and integration testing and provide results in a consolidated format. Thresholds are defined and maintained for each project to create gates for promotion and prevent unwanted code to deploy to environments.
 - Jenkins Dashboards preconfigured with Test Coverage Plugins
 - Enforcement of Code quality and testing thresholds to deployment and promotion gates
- **Scan** - Our team has created the Seed Pak which contains a starting point for your application development. This includes the ability to add gates and thresholds to your application. The pipelines will support for Software Composition Analysis (SCA) and Static Analysis Security Testing (SAST) as part of the build definitions. Data is published and stored over time to allow for analysis of development trends.
 - SAST scanning over time with SonarQube
 - Java SAST scanning within maven builds to include Spotbugs and PMD
 - Java Code Formatting checks using Checkstyles
 - JavaScript Linting
 - Container Security Scanning
- **Deliver** - Our team has created a standardized repository packaging format with tooling and integration points that enable deployments of applications, container images, and artifacts. We have also tied this

into tooling and integration for security scanning of any published container images. This allows us to scan prior to delivery, publish and promote current artifact and image repository to public repositories, including but not limited to Maven Central and Dockerhub.

- Artifact Repository Installation and configuration
- Docker Image Repository Installation and configuration

4.8.2. Continuous Deployment (CD)

Figure 4.14. Framework for Continuous Deployment



- **Deploy** - Titan provides simple pathways to execute deployments using standard branching, meaning that the code will always represent what is deployed.
 - Titan Deploy allows teams to execute deployments without knowing anything about target Kubernetes environment but the name.
- **Promote** - Titan provides simple pathways to execute promotions using standard branching, meaning that the code will always represent what is deployed.
 - Titan utilizes Argo Rollouts to give power to teams to allow them to define how they would like to promote, be that via simple replacement, Canary Deployments or Blue Green Deployments.
 - Titan Deploy allows teams to promote without knowing anything about the target environment but the name.
- **Operate** - Fix problems in production by giving teams just the right permissions to deploy and maintain applications in each environment.
 - Kubernetes Role Based Access Controls (RBAC) restricts all users from directly publishing artifacts to the cluster, enforcing that nothing is deployed without committing code for traceability.
 - Creating Team-based spaces allows teams to perform necessary functions within their own teams without impacting other team's deployments.
- **Monitor** - Give operational analytics and logs to the users via Dashboards.
- **Respond** - With our deployment workflows, we can easily manage promotions to your cluster. If there are defined health thresholds, we can roll back the promotion to the last known good state.

4.8.3. Packaging and Distribution

Teams will be targeting Java archive (JAR) distributions to maven central. These will be created via build server from an internal version control repository (VCS). This code will be published to a public VCS (HL7 GitHub). The java packages (jars, wars, etc) will be published to the local artifact repository, then promoted to Sonatype via a managed sonatype account. This will then promote to maven central as part of the release process.

With installed client software, installation packages will be created to download and package the installation for execution. We will approach first installations directly on Windows using a Microsoft Installer (MSI) and MacOS using a Disk Image (DMG) installation. After this, we will discuss possibilities of other platforms and package formats, such as deploying on Linux servers, virtual machine images, and container images. Based on the work, we will prioritize this based on customer feedback. Initial software installation packages have already been created and are being tested.

Web Applications on the reference implementation will be targeting Linux installations using containerized deployments. For initial releases, we will test this in Kubernetes and distribute for use on a Docker-Hub account.

With any of these distribution packages, our long-term goal is to set up maintainable processes that allow us to test them on a variety of different setups before committing to distribution. These processes will be captured in the code as documentation to the projects, making it easy for new public members and teams to contribute to the process and codebase, as well as manage feedback.

4.8.4. Expected Changes to CI/CD

As we build out the CI/CD processes, we will regularly re-evaluate our tooling, as we want to move to a solution that best can be executed and delivered with contributors that will not be able to access private environments and use private licenses.

The first of these is the move from the private hosted Atlassian JIRA solution to a more sustainable Atlassian JIRA SaaS, which can be accessed by the community and contributors can provide regular feedback.

Other pieces will be more difficult, as security and license scanning require licensing and will need to have gatekeeping in place to not over-extend the project, financially.

4.9. Scalability

With the analysis of the current state, the knowledge management platform is set up to allow for some concurrency in use. This, however, needs to be tested to validate that no corruption of data occurs, and that expected load can be maintained. Performance testing will need to start with web access first, followed by direct access from users. Performance service level agreements (SLAs) will need to be further flushed out and defined, and automated performance tests will need to be created to execute at each sprint end, prior to a release. As currently no part of this exists, this will be something we focus on in the architecture and design discussions.

4.10. License

As we prepare to make the products available with installers, teams will need to do a thorough analysis of licenses and attribution requirements. As the project plans on only using open-source licenses to reduce cost and increase adoption, we must maintain that we are compliant with this requirement and the different open-source licenses. This may require interfacing with legal teams and use of license scanning to ensure no conflicts of purpose.

4.11. Security

As part of the CI processes, we will integrate Security scanning. Security scans will be applied so that users can be sure that code that is installed will not have adverse effects on the installed computer. Also, it needs to ensure that bad actors don't get access to data that they should not be privileged to. To accomplish this, our team will need to include several forms of SAST and Dynamic Application Security Testing (DAST).

SAST helps to analyze a software program without executing the program, providing security scanning and defect scanning. Comparing with dynamic analysis, static analysis analyses the exact source code line while dynamic analysis analyses the program while it is being executed; by performing analysis of all kinds of tests on the object codes and executable. In summary, static analysis tools help:

- detect possible bugs which the debugger is unable to detect.
- provide vulnerability and remediation information.

DAST helps to analyze a software program while the program is executing, determining if there are runtime issues. To do this, our team will need to develop methods for testing against multiple Operating Systems as well as execute regular penetration and vulnerability tests.

4.12. Privacy Needs

As we further develop this reference implementation that may contain sensitive information, with things like Health Insurance Portability and Accountability Act (HIPPA) protected information and Personally Identifiable Information (PII), it becomes very important that this data protected. Communications will need to be designed to require encryption in transit. Services will need to be designed to require encryption at rest. To share any data, there will need to be designed sharing patterns that either strip information of identifying information and aggregate it with other information, or it will need to facilitate data sharing agreements as part of the sharing process.

This is a complex topic, and we will need to bring in security experts as part of this process to make sure that this is maintained. Also, we will need to have notification systems in place to cover zero-day issues and allow us to regularly update the codebase to address security concerns as the product matures.